Aalto University
School of Science
Master's Programme in ICT Innovation

Kuldeep Singh

Survey of NoSQL Database Engines for Big Data

Master's Thesis
Espoo, May 11, 2015

Supervisor: Assoc. Prof. Keijo Heljanko
Advisor: Olli Luukkonen, D.Sc. (Tech.)

Aalto University
School of Science
Master's Programme in ICT Innovation

ABSTRACT OF
MASTER'S THESIS

Author: Kuldeep Singh

Title:
Survey of NoSQL Database Engines for Big Data

Date: May 11, 2015 **Pages:** 83

Professorship: Department of Computer Science **Code:** T-110

Supervisor: Assoc. Prof. Keijo Heljanko

Advisor: Olli Luukkonen, D.Sc. (Tech.)

Cloud computing is a paradigm shift that provides computing over Internet. With growing outreach of Internet in the lives of people, everyday large volume of data is generated from different sources such as cellphones, electronic gadgets, e–commerce transactions, social media, and sensors. Eventually, the size of generated data is so large that it is also referred as Big Data. Companies harvesting business opportunities in digital world need to invest their budget and time to scale their IT infrastructure for the expansion of their businesses. The traditional relational databases have limitations in scaling for large Internet scale distributed systems. To store rapidly expanding high volume Big Data efficiently, NoSQL datastores have been developed as an alternative solution to the relational databases.

The purpose of this thesis is to provide a holistic overview of different NoSQL datastores. We cover different fundamental principles supporting the NoSQL datastore development. Many NoSQL datastores have specific and exclusive features and properties. They also differ in their architecture, data model, storage system, and fault tolerance abilities. This thesis describes different aspects of few NoSQL datastores in detail.

The thesis also covers the experiments to evaluate and compare the performance of different NoSQL datastores on a distributed cluster. In the scope of this thesis, HBase, Cassandra, MongoDB, and Riak are four NoSQL datastores selected for the benchmarking experiments.

Keywords: Big Data, BASE, CAP, Cloud Computing, NoSQL, Cassandra, Cockroach DB, HBase, MongoDB, Riak

Language: English

Acknowledgements

This work would not have been completed without help and support of many individuals. I am grateful to my supervisor Keijo Heljanko for providing me an opportunity to do my thesis under his supervision. This thesis could not be completed without his helpful comments and guidance. I am grateful to Aalto University for giving me the chance of finishing my studies in Finland and for the opportunity of using Triton cluster for this thesis. I would like to extend my thanks to Olli Luukkonen for his valuable advice. I could not be able to finish the thesis without the blessings of my lord Tirupati Balaji and support of my friends: Yanka, Achal, Rui, and Kai.

I dedicate this thesis to three pillars of my life: my parents, my teachers, and Mr. Rakshpal Singh Chauhan.

Espoo, May 11, 2015

Kuldeep Singh

Abbreviations and Acronyms

ACID	Atomicity, Consistency, Isolation and Durability
API	Application programming interface
AWS	Amazon Web Service
CAP Theorem	Consistency, Availability, and Partition Tolerance Theorem
CRDT	Commutative Replicated Data Types
DBMS	Database Management System
DHT	Distributed Hash Table
GFS	Google File System
HDFS	Hadoop Distributed File System
HTML	HyperText Markup Language
HTTP	The Hypertext Transfer Protocol
IT	Information Technology
JSON	JavaScript Object Notation
LSM Tree	Log–Structured Merge Tree
MVCC	Multiversion Concurrency Control
NIST	National Institute of Standards and Technology
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
PDF	Portable Document Format
RDBMS	Relational Database Management System
SEDA	Storage Event Driven Architecture
SQL	Structured Query Language
TCP	Transmission Control Protocol
WAL	Write–Ahead Log
XML	Extensible Markup Language
YCSB	Yahoo! Cloud Serving Benchmark

Contents

List of Figures

Chapter 1

Introduction

We are living in the digital world. Last two decades have seen significant growth in the information on Internet. Computers, cellphones and even physical devices such as sensors are now connected to Internet. With the rapid outreach of Internet in the lives of people, it is a need of the hour to focus on the technological advancements which can manage large amount of data and allow easy access. Cloud computing is one of such breakthrough technologies in the virtual world.

1.1 Cloud Computing

Late Steve Jobs, former CEO, Apple Inc. whenever appeared on any stage, everyone was keen to know what new product he may introduce to the world. Apple is the company which made the gadgets like Macintosh, iPhone and iPod. In 2011, he appeared on one of the stages but that time he did not show any gadget. He only talked about a service that people can use to keep their personal data and information in sync on all Apple devices. He talked about iCloud. iCloud demonstrates the strength of cloud computing and Apple just leveraged that successfully.

It does not matter what is the name of the service or who is the service provider such as Apple, Microsoft or Google but it is a belief that cloud computing is the future. Many tech gurus assure that it will change the lives of people. Big Information Technology (IT) companies are now forecasting that cloud computing will play a major role in defining IT business solutions.

Cloud computing in reality is an Internet based mechanism and a facility to use computer applications. Google Apps is an example of cloud computing which provides business applications online. The online information is stored in servers permanently. The information is also stored in user's desk-

top, cell phone, gaming console, and on other devices temporarily. Cloud computing enables users not only just store information on their own machines but also in remote cloud databases. Users can access all information available on Internet and the different applications through web services. Also, different computing resources such as network, CPU, storage, and applications are made available to the users by cloud providers at minimum costs. Cloud providers sell resources elastically and users can pay–as–you–go billing system. Cloud computing has further helped to reduce initial financial requirement of running new IT service or business because of the low cost infrastructure provided by the cloud operators.

The National Institute of Standards and Technology (NIST) [73] provides formal definition of cloud computing as "model for enabling ubiquitous, convenient, on–demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." Cloud model consists of five essential characteristics, three service models, and four deployment models [73].

1.1.1 Essential Characteristics

There are 5 essential characteristics of cloud model:

- On–Demand Self Service

  On–demand self service refers to a facility provided by the service providers to the customers. Customers can access the cloud resources online whenever they require. Cloud computing enables convenient on–demand network access to the shared pool of computing services such as email, chat application, network resources etc. without much direct live interaction with the vendors.

- Broad Network Access

  A major advantage of cloud computing is that user can access the resources via wide range of networks. It can be accessed over phone, computer, tablet, and workstations.

- Rapid Elasticity

  Cloud computing enables users to expand their computing resources very easily. IT resources can be either scale out or scale up based on customer's need. With the growing business or in order to process

increasing amount of data, the need for parallelism of computing resources arises. It helps to achieve higher processing speed to provide faster service.

According to Moore's law, the number of transistor in an integrated circuit doubles in every 18-24 months [78]. In 2005, Herb Sutter in his research stated that the clock speeds of microprocessors are not going to improve much in the near future and the number of transistors in a microprocessor is increasing with a high rate. Because of this, there is a need to change programming models to exploit all the available concurrency [89].

Scalability of processors is one of the major focus now. Scaling–up and scaling–out are two techniques to scale the resources [76]. When the need of scaling arises, a single powerful computer with more core and more memory is added to provide parallelism. This is known as ***scaling–up***. In contrast, when the need of parallelism pops–up, instead of adding a single powerful computer many small computer with relatively slow processors and moderate memory is added to the system. This is known as ***scaling–out***. Cloud computing leverages the scaling out techniques to provide elasticity to the users. It gives service providers an advantage in case if they want to expand their services. Cloud companies provide elasticity to their users by automating most of their work and using scalable software, advance distributed technologies, automatic fault detection, and recovery.

- Resource Pooling

  The service providers allow multiple tenants and application to use resources on a shared basis. Resource pooling helps to provide cheaper services to the customers. Resources can be either physical or virtual and can be reassigned or dynamically assigned based on the need of a particular customer.

- Measured Service

  This characteristic is a speciality of cloud computing. Users can manage and control the amount of service utilization. There is metering capability for cloud computing resources which can be used to optimize its use. It is similar to utility services like electricity, water or postpaid telephone. The cloud services are also available on prepaid or postpaid contracts. User can choose from different pricing options available.

1.1.2 Deployment Models

For the management of cloud services, there are four types of deployment models.

- Public Cloud

  Public cloud is the most widely used deployment model. In this, a company depends on third party for cloud services. The cloud services can be used by users based on pay–what–you–use manner. The deployment of public cloud is very fast and it is easily scalable. Amazon, Microsoft, and Google are some of the main public cloud providers.

- Private Cloud

  It is a type of cloud which is solely maintained, operated and owned by a single organization. Private clouds are mostly deployed by big organizations, government institution or universities within their firewalls.

- Hybrid Cloud

  Hybrid cloud is a mix of public and private cloud. It gives an option to the organizations to keep some of their data privately on cloud and rest as public for others. It utilizes the cost saving of public cloud with security and privacy of private clouds.

- Community Cloud

  Community cloud is used by the organizations those are doing similar kind of business or have similar requirements. This type of cloud model is not really public, but bit more open compared to private cloud when it comes to sharing. It gives an advantage to the organizations to share costs in case of scaling further.

1.1.3 Service Model

Cloud vendors offer resources in three forms. These are: Software as Service (SaaS), Platform as Service (PaaS), and Infrastructure as Service (IaaS).

1. Software as Service (SaaS)

   SaaS enables the cloud users to use cloud vendor's software application via Internet and it run on a cloud infrastructure. User need not to worry about purchasing or maintenance of the infrastructure. These applications can easily accessible through service oriented architecture and web

base interfaces. SaaS has become a prominent model for many business support applications like management information system (MIS), accounting software, Customer relationship management (CRM) and Enterprise Resource Planning (ERP). It follows freemium model for payment. In freemium model [55], limited functionality is available for free and for the premium functionality, user need to pay some charges. For this, online payment details must be provided for billing purposes.

The use of multi–tenant architecture makes SaaS application available at very low price due to sharing of resources and infrastructure.

Dropbox [19] is an online backup service which is based on a SaaS solution. It is developed by Dropbox Inc. and launched in September, 2008. Dropbox allows users to store their personal data on cloud, file synchronisation and even have their own personal cloud. Dropbox is offered in five different languages and have customers in over 175 countries [18]. TechCrunch named Dropbox start–up of the year in 2012 [51].

Just like Dropbox, Flickr [35] is another SaaS solution which is mainly for picture storage. Flickr is owned by Yahoo. In 20011, Flickr said that they have over 6 billion stored images and this number is growing continuously.

CRM is the largest market for SaaS. Its revenue was forecasted to reach 3.8bn US dollar (USD) in 2011 [40]. Salesforce.com [32] is a popular CRM platform. The company was founded in 1999 and has revenue over 4bn USD in 2014. Sales Cloud and Service Cloud are two major products of Salesforce. Sales Cloud is dedicated to manage sales and associated business whereas Service Cloud is a multichannel help desk which allows customers to build communities for mutual help.

2. Platform as Service (PaaS)

   PaaS comprises an environment for the deployment and development of cloud applications. The vendors provide platform which allow users to be free from maintenance and management of the infrastructure. Users can directly run and manage web application without much complexity.

   Force.com is a platform developed by Salesfoce.com. It allows users to build an app, connect it to any other platform and manage it from any location. Force.com has inbuilt tools and services for making apps. Software as service of Salesforce.com uses Force.com as its platform. Customer need not to worry about platform as he/she only has control over deployment and configuration of the applications on Force.com.

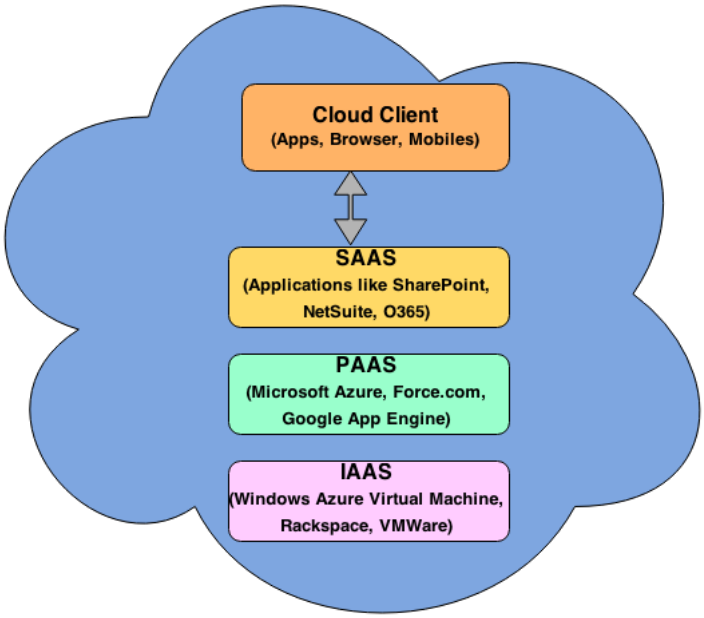

Figure 1.1: Cloud Service Models

Openshift [28] is also a PaaS product. It is developed by Red Hat under Apache license 2.0. It accelerates the IT service delivery and allows users to host their applications in public cloud. A considerable amount of automation is used in Openshift which gives an edge to the users to focus only on their development and creativity. It supports Java, JavaScript, .NET, Perl and many other language environments for flexible development.

Heroku [20] is an another cloud Platform as Service. Initially it only supported Ruby programming environment but over the last couple of years, it also supports several other programming languages like Java, .NET, Python, NodeJS, and Scala. Saleforce bought it in 2008. Heroku is very easy to use. It provides documentation for configuration and running of the applications. It is also powered with additional support from add–ons and can be easily integrated with GitHub.

3. Infrastructure as Service (IaaS)

   The users rent the cloud infrastructure such as storage, network and servers from the cloud vendors and access it over Internet. There are various examples of enterprise usage of IaaS such as enterprise infras-

tructure, cloud hosting and virtual data centers. Amazon Web Service (AWS) [2] is the leading IaaS provider. Amazon launched it in 2006 and user can access it via HTTP or over REST or SOAP protocol. Amazon web services offer variety or products ranging from compute, networking, content delivery, database, deployment and management.

Microsoft Azure [21] is a combined PaaS and IaaS product. Azure was commercially released in 2010. Azure provides three different services. The first is Windows Azure Websites, which is a Platform as Service product of Microsoft. It allows user to host websites built in different programming languages like PHP, ASP.NET, and NodeJS. The second is Windows Azure Cloud Service which is also a PaaS product. They are the containers for the application deployed by the users. The hosted application can easily integrated with external web applications such as e-commerce websites.

Another example is Virtual machine. It is an IaaS product offering. Users can make their own virtual machines and have full access over Microsoft data centers. It supports both Windows and Linux server images.

1.1.4 Cloud Computing Technologies

Cloud computing has a base of wide variety of technologies and fundamental concepts. There are many basic building blocks of cloud computing that collectively and collaboratively synchronize together to build a model for on–demand access to shared pool of configurable computing resources. Let us now discuss some of its building blocks in brief.

- Virtualization of Computing Resources

  Cloud computing leverages the advantages of virtualization techniques. Cloud computing is not same as virtualization. Virtualization is the abstraction of computing resources with its own storage and network connectivity. Cloud computing uses virtual resources and determines its allocation and usage [39]. Cloud computing needs virtualization but not vise–versa. There are many examples of virtualization in cloud computing technologies. Some of them are Amazon EC2 [1], OpenNebula [27], and OpenStack Compute [29].

- Scalable File Storage

  With the growing size of information over Internet, there is a need of scalable file storage for storing large number of files and their fast

access. Many scalable file storage systems such as Amazon S3 [3], Google file system (GFS) [84], Hadoop Distributed File System (HDFS) [36] provide scalable storage of files for cloud users and providers.

- Scalable Data Storage

  Many a times, term "NoSQL datastores" is used for the database systems that do not fully support all features of traditional relational databases. A new term "NoSQL" is used for the systems which do not support SQL [46]. NoSQL means "not only SQL". We will discuss few NoSQL datastores in next few sections. Some of the NoSQL datastore are HBase, Cassandra, Redis, Riak, CouchDB, and MongoDB.

- Existing Web Technologies

  Cloud computing also consists of existing web technologies. Several web standards like HTTP, HTML, DHTML, XMPP, SSL, and OpenID are part of cloud computing technologies. Besides this, web services, such as REST, SOAP, and JSON also play an important role in developing basic building blocks of cloud based applications [90].

1.2 Big Data

With the growing outreach of Internet in the lives of common people, as of 2012, 2.5 exabytes of data were created per day in digital world. Such large amount of data is difficult to process using traditional data processing techniques. These data sets are often refer as "Big Data". Every day new data is generated from video, pictures, sensors, social media etc. Often, huge volume of data is messed up and it is a necessity to find new ways to store and process data. For processing large amount of data, hard disk parallelism is one of the goal for scalable batch processing. A programming paradigm MapReduce is developed by Google for scalable batch processing.

1.2.1 MapReduce

Google MapReduce [56] was initially used for web search indexes. Google is an Internet giant. It needs hard disk parallelism to store and process big amount of data. MapReduce is developed for batch processing and it can run on multiple machines in parallel. It also increases the productivity of the programmers because once MapReduce is developed, programmers need not to develop individual tool for taking advantages of hard disk parallelism.

The programmers need not to worry about fault tolerance, parallelism, load balancing and synchronization. MapReduce takes care of everything.

MapReduce uses concepts of functional programming [65]. Map and Reduce are two side–effect free function in MapReduce framework [71]. In case of fault tolerance, re–execution of the functions is used. For example, in a distributed cluster if a node is failed, a Map or a Reduce task can be re-executed on different nodes. The output is same and it is independent of the number of nodes used for computing.

"Mapping a list" and "Reducing a list" are two processing concepts in functional programming. In Mapping a list, each list element is processed with a mapping function to get the mapped list elements as output. Then, a reduce function is applied on mapped list elements in sequential order to get a final output. Key–value pairs are the output of map function in MapReduce framework. Map function takes raw block of data of size 64-128 Mb and produce key–value pairs. The map output is grouped together by MapReduce framework like (key, (list of values)). It is the format of all values with same key grouped together. This acts as an input to reduce function. User only provides map and reduce function in the MapReduce framework. A special master node re–executes map and reduce function for scheduling and fault tolerance. The master is also responsible for creating a particular number of map and reduce workers. Master then splits work for them. In MapReduce, nodes can only communicate via shuffling from mapper to reducer. In shuffling, intermediate key–value pairs are exchanged. Figure 1.2 explains the complete MapReduce process.

Open source clone of Google MapReduce is Apache Hadoop [6]. It is developed by Yahoo. Apache Hadoop has its own distributed file system known as Hadoop Distributed File System (HDFS). HDFS is inspired by Google File System (GFS). Apache MapReduce is a module of Apache Hadoop based on Google MapReduce. Apache Yarn is an another module which is used for job scheduling and cluster resource management in Apache Hadoop. For last many years, Hadoop is one of the top Apache Software Foundation projects.

The field of cloud computing is expanding and everyday immense amount of data is generated by many different sources. This large quantity of data is essentially needed to be processed and stored effectively in the databases. For initial years, traditional Relational Database Management Systems (RDBMS) were fullfilling the need to store large amount of data. Due to increase in online data and increasing number of Web applications, the requirements have been changed over time. Traditional databases were unable to scale as per requirement and could not met the growing needs to scale effectively on large distributed clusters.

To process and store petabytes of data, Google developed BigTable [47],

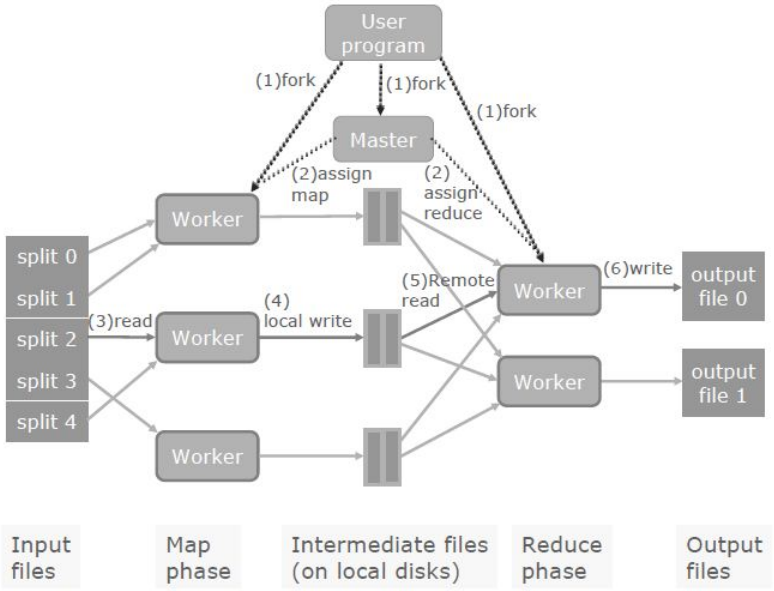

Figure 1.2: MapReduce Process
[56]

which is a distributed storage system for structured data. It is build on top of the Google File System. BigTable is a highly available, scalable and high performance datastore. Apache Software Foundation created open source version of BigTable and it is known as Apache HBase [7]. Slowly Apache HBase gained popularity and soon became one of the major projects of Apache. HBase uses HDFS as its storage system and Zookeeper as its coordination service. The design of Zookeeper is inspired from Google Chubby [41]. The Chubby is a lock service for loosely-coupled distributed systems.

Over the recent years, many different databases have been developed to store massive cloud data. These databases are called NoSQL datastores. NoSQL datastores are distributed databases developed to offer high performance and scalability while dealing with Big Data. This thesis covers a holistic overview of few NoSQL datastores.

1.3 Structure of the Thesis

The Thesis is organized in 9 chapters. Chapter 2 is about the background of the thesis and describes how database technologies have evolved over the years. In Chapter 3 we discuss document Datastores and its proper-

ties. Chapter 4 and Chapter 5 describe Extensible Record Datastores and Key–Value Datastores respectively. We cover different aspects of these datastores such as architecture, data model, storage structure, properties and fault tolerance support. Chapter 7 covers environment and benchmarking setting for datastores. Chapter 8 illustrates different benchmarking results of these datastores. Finally, Chapter 9 concludes the thesis with future references.

Chapter 2

Background

This chapter gives brief introduction about the journey from SQL databases to NoSQL datastores. The chapter also highlights important concepts and fundamentals related to NoSQL datastore development.

2.1 Evolution of Databases

A database is a collection of data items that provides an organizational structure for information storage [88]. Database also provides a mechanism for querying, creating, modifying and deleting data. A list can also be used to store data but in a list, redundancy is a major issue. A database can store relationships and data that are more complicated than a simple list with lesser or no redundancy.

A relational database stores data in tables. Normally a table is based on one information theme. For example, an employee list can be divided into manager table, intern table, and junior staff table. A table is a two dimensional grid of data that contains columns and rows. The convention in relational database world is that columns represent different attributes of an entity and each row represents the instance of the entity.

Conceptually, database is a component of database system. Besides

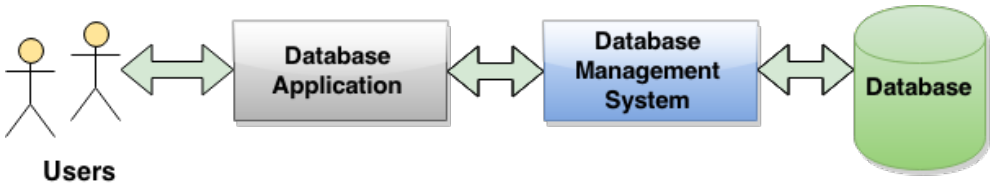

Figure 2.1: A Database System

database, database system consists of database users, database applications and Database Management Systems (DBMS). Database users need not to be always human. It is possible, for example, for other software programs to be users of the database. Users interact with database application and application further depends on the DBMS to extract and store data in the database. The DBMS acts as a gatekeeper. All the information flowing in or out of database must pass through the DBMS. It is a critical mechanism for maintaining quality of data and database. Users and database applications are not allowed directly to interact with database. The Figure 2.1 illustrates a typical database system.

The database systems are based on some data models. Data models describe the logical structure of data items and their associated operations. Create, update, read and delete operations are are four basic ways to interact with a database. They are often known as CRUD operations [81]. The Structured Query Language (SQL) is a standard way to execute CRUD operations on database.

A Database Management System is an intermediary between database applications and database. The DBMS creates and manages the database. DBMS can be categorized based on its data model. Relational Database Management Systems (RDBMS) [50] use relational data model [49]. Dr. E.F. Codd developed RDBMS in 1980s. RDBMS maintain data in tables and relationships which are created among data and tables. Database is divided into tables and they are connected through a "key field". RDBMS is the most famous and used database model.

Over last four decades, RDBMS remain a key technology to store structured data. But with growing size of data, companies do need modern technologies to maintain and process data. RDBMS are not that good for large data volumes with varying datatypes. They also have scalability problem and often result into failure while performing distributed sharding. Oracle Real Application Clusters (RAC) [30] is a relational database cluster that provides high availability, reliability and performance. Also, MySQL cluster [24] is another example where relational databases scale on large cluster. RDBMS satisfy ACID (Atomicity, Consistency, Isolation and Durability) properties defined by Jim Gray in the late 1970s [62]. Consistency is bottleneck for scalability of relational databases. RDBMS follow strict data model and can not violate ACID properties. That is why NoSQL datastores were developed to address the challenges of traditional databases.

In 1998, Carlo Strozzi used the term "NoSQL" first time [26]. Rick Cattell explains NoSQL as "Not Only SQL" [46]. NoSQL datastores have weaker concurrency model than RDBMS. They often do not support full ACID guarantees. Their ability to scale horizontally and high availability has given wide

acceptance to NoSQL datastores in recent years. Especially in the cloud based companies and service providers, NoSQL datastores are hugely popular. Now NoSQL datastores have wide acceptance in variety of industries ranging from manufacturing, oil and gas, energy, banking and health care.

2.1.1 Types of NoSQL Datastores

There are three main types of NoSQL datastores: *Key–Value Datastores*, *Extensible Record Datastores* and *Document Datastores* [46].

- In the Key–Value datastores, the values are indexed with keys and its data model follows a famous memcached distributed in–memory cache [46]. Examples include Project Voldemort, Riak and Redis.
- Document datastores retrieve, manage and store semi structured data. They provide support for multiple form of documents (object). The values are stored in documents as lists or nested documents. Few examples are MongoDB, SimpleDB, and CouchDB.
- Extensible Record datastores are motivated from Google's BigTable. We will discuss BigTable briefly further in this chapter. It has flexible data model with rows and columns. Rows and columns can split over multiple nodes. HBase, HyperTable, and PNUTS are its few examples.

2.2 The Foundations of NoSQL Datastores

NoSQL datastores have three foundation principles: the Consistency Model, the CAP Theorem and BASE property [92]. We discuss them in the following section.

2.2.1 CAP Theorem

In 2000, Eric Brewer introduced the CAP theorem [43]. Two years later, Gilbert and Lynch formally proved the CAP theorem [61]. This theorem states that for any distributed datastore, there are three basic properties with interdependency. These properties are:

- **Consistency**: It means that data remain consistent in the database after each operation.
- **Availability**: It means the database system is always available with no downtime. Irrespective of success or failure, every request receives a response.

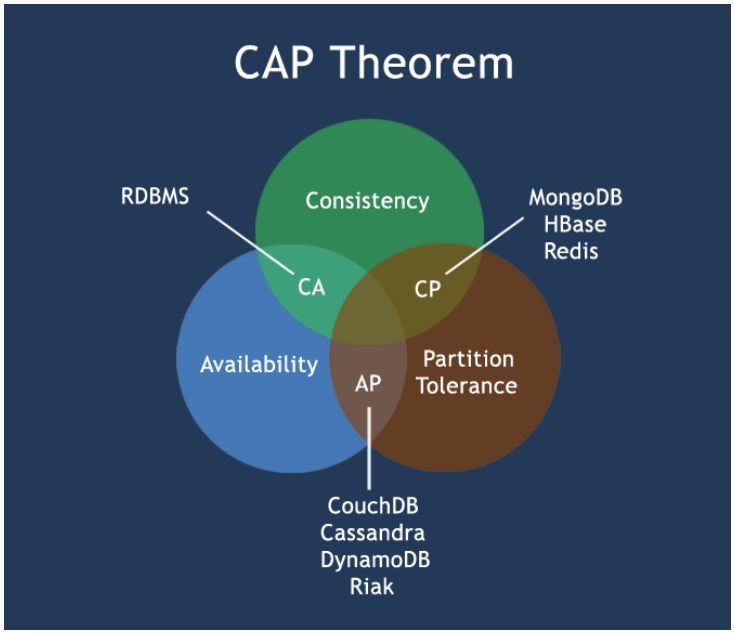

Figure 2.2: The CAP Theorem [14]

- **Partition Tolerance**: It means that in case of network partitioning, the system continues to function. Even if the communication between different database nodes in the network is unreliable, everything works.

According to the CAP theorem, it is not possible for any distributed database to satisfy all three properties at a time. Therefore, distributed databases can only satisfy at most two of these three properties. Logically, a distributed system must satisfy Partition Tolerance condition otherwise a database can not be considered as distributed. Hence any distributed database needs to choose either Consistency or Availability along with Partition Tolerance.

The Figure 2.3 describes different possible trade–offs as per CAP theorem. These are primarily three types of datastore design:

- **CA Systems**

  These are non distributed database systems which are consistent and available but not partition tolerant. Few examples include relational databases, HDFS NameNode, Vertica and Aster Data. These systems are easy to program and implement but have limited scalability. They are preferred in use when system load is low.

- **CP Systems**

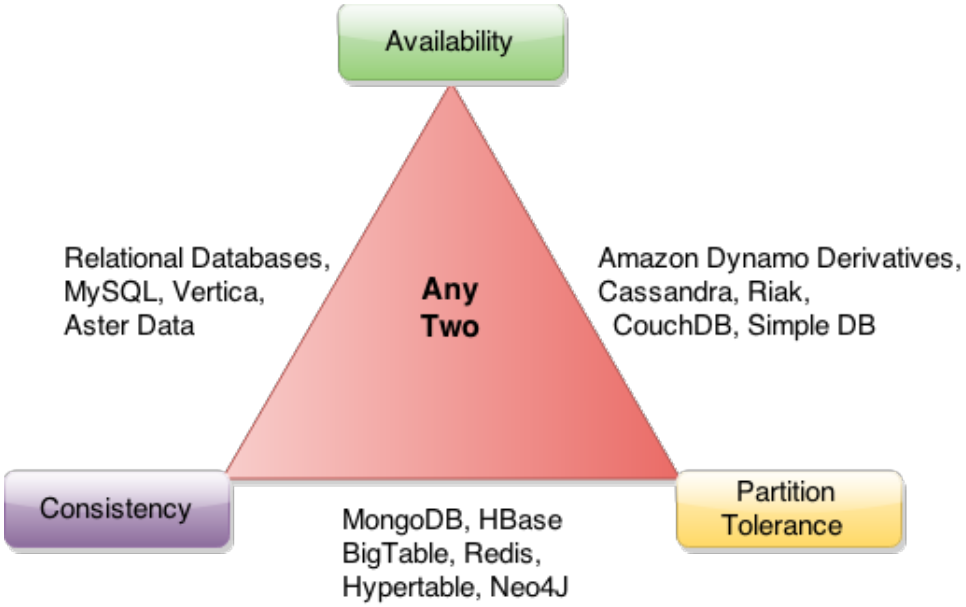

Figure 2.3: The CAP Theorem Trade–Offs

These distributed datastores are consistent and partition tolerant but compromise availability. The datastores can not be altered when network splits into partitions. Example includes HBase, BigTable and its all derivatives, Redis and Neo4J. They are easy to program and achieve data consistency in scalable way. They have high latency for updates. It is claimed by many that the MongoDB is a CP system. A recent blog post bursts this popular notion and proves that the MongoDB does not always follow CP system behavior [12]. MongoDB consistency model is broken by design and it has few bugs [1] [2] .

- **AP Systems**

  These systems and available and partition tolerant but not always consistent. They can become inconsistent when network splits into partition. Some examples include Amazon Dynamo derivatives, Cassandra, Riak, CouchDB, and Simple DB. These distributed database systems are not easy to program but provide extremely good scalability. They are good for low latency applications.

[1] MongoDB bug: https://jira.mongodb.org/browse/SERVER-17975 (accessed: 11/05/2015)

[2] MongoDB bug: https://jira.mongodb.org/browse/SERVER-18022 (accessed: 11/05/2015)

2.2.2 BASE Property

The NoSQL datastores do not provide ACID transactional guarantees. They follow relatively weaker consistency model. BASE [80] stands for "Basically Available, Soft state, Eventually consistent". BASE brings a softer consistency model. *Basically Available* means the datastores assure system availability in terms of CAP theorem. *Soft State* states that the system state may change over period of time even if no input is given. Finally *Eventual Consistent* indicates that the system eventually become consistent with time if system is not feed with any input during that time. These datastores give priority to availability over consistency i.e. they are AP systems. Amazon Dynamo and Apache Cassandra are examples of such systems.

2.2.3 The Consistency Model

After Eric Brewer introduced CAP theorem, the different trade–offs between Consistency and Availability have been studied [44]. The client and the server have different aspects for consistency. There are three types of Client side consistency desribed by Werner Vogels [91]:

- Strong Consistency: It refers to the situation when read operation returns the most recent written values.
- Eventual Consistency: It means that all updates will reach to all nodes of the distributed datastores but it will take some time. Therefore, all nodes will eventually have same value.
- Weak Consistency: There is no assurance that the subsequent access to the distributed datastores will return same value.

On the server side, consistency is concerned with server updates. NoSQL datastores offer some or even total consistency. HBase, BigTable are strongly consistent datastores whereas Apache Cassandra is eventually consistent.

2.3 BigTable

The development of BigTable at Google was a breakthrough in the field of cloud datastores. The fundamentals and concepts used in the BigTable design are now widely used in many NoSQL datastores. Therefore it is important to have a brief look into BigTable design and properties.

BigTable is a highly scalable, consistent and partition tolerant system. It uses Google File system as base storage platform. In BigTable, writes are

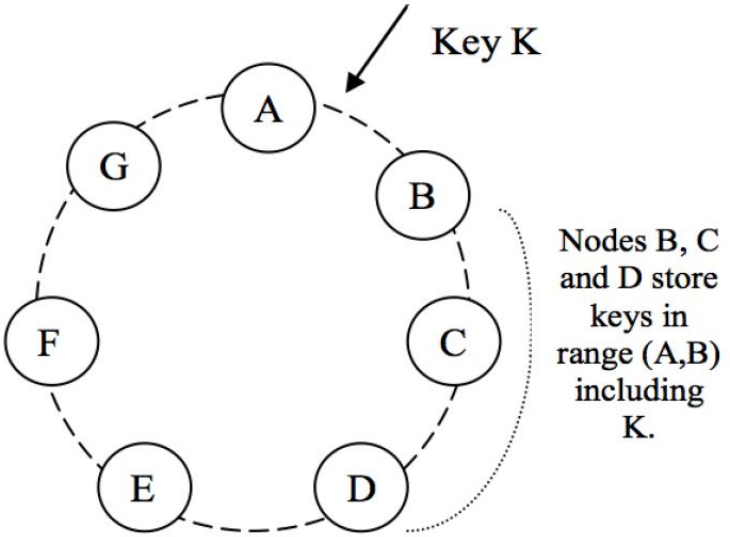

Figure 2.4: Consistent Hashing with Three-Way Replication in Apache Dynamo [57]

optimized over reads. Only sequential reads are possible in BigTable that increases the write speed.

BigTable has a distributed, sparse multi–dimensional data model. The data is stored as strings and it can be indexed by three coordinates, namely a row, a column and a timestamp. Hence, a map is created using these three indexes [47]. A row key is used to sort data and BigTable automatically shard the data across multiple servers. On the other hand, *column families* are used to group columns. Timestamp is used for version control mechanism of data. Though BigTable does not assures multi–row ACID guarantees but updates on a single row is atomic. Tablet is the component of BigTable design that is used for row key distribution and load balancing. A fixed data structure is used to store BigTable data files. This is known as SSTable. BigTable heavily use compaction when SSTable grows in number. The compaction idea is inherited from the paper "The Log–Structured Merge–Tree (LSM–Tree)" [79].

A lock service known as Chubby [45], is a fault tolerant distributed coordination system. BigTable relies on it for coordination between different nodes.

2.4 Amazon Dynamo

Amazon Dynamo [57] is the parent of all eventually consistent NoSQL datastores. It is an AP system. It is a key–value store and uses Distributed Hash Table (DHT) [93] to distribute the contents of hash table. Consistent hashing [68] allows natural elasticity to add or remove more servers in a distributed datastore system. While distributing data over nodes, Dynamo does three–way replication. It means two more copies of each stored data is present in the system.

Dynamo allocates a version number to each data item. Dynamo uses data versioning to implement the eventual consistency model [57]. Dynamo also uses vector clocks to store several version of data simultaneously in the datastore. For further details and other relative concepts, refer to the Amazon's Dynamo Paper [57].

2.4.1 Features of NoSQL Data Stores

Rick Cattell listed many features of NoSQL datastores in his paper "Scalable SQL and NoSQL Data Stores". We will list some of the features that are common to many NoSQL datastores:

- NoSQL datastores has ability to scale horizontally over many nodes.
- They can easily replicate and partition data over multiple nodes.
- Their concurrency model is often weaker then ACID.
- RAM and distributed indexes are very well used in NoSQL datastores.
- The data schema is flexible enough to add new attributes to the data.
- Many NoSQL datastores have inbuilt support for automatic sharding unlike manual sharding in case of traditional RDBMS.
- Several features of SQL such as join and global transactions are not supported by NoSQL datastores.
- NoSQL datastores have sparse data model unlike RDBMS.

These properties are generalized features of NoSQL. Many NoSQL datastores have specific and exclusive features and properties. They also differ in their architecture, data model, storage system and fault tolerance abilities. We will discuss different NoSQL datastores in next couple of sections in detail.

Chapter 3

Document Datastores

Document Oriented datastores are design to store, retrieve and manage semi structured data. They support multiple type of documents (objects) per datastores. "Documents" save values as nested documents or lists. These documents are of any type ranging from PDF, Word document, XML, HTML, etc. SimpleDB, CouchDB, and MongoDB are few examples of Document Oriented datastore. This section covers the design principles of MongoDB.

3.1 MongoDB

MongoDB [23] is an open source, document oriented datastore that is written in C++. It is developed by 10gen (Now MongoDB Inc.) for a wide variety of real time applications. It also provides full index support for collection of documents. MongoDB has a well structured document query mechanism. Next few subsections discuss different aspects of MongoDB design.

3.1.1 Architecture

NoSQL datastores are quite handy to deal with much large velocity and volume of data. MongoDB is a scalable and high performance NoSQL datastore. It is an agile datastore that allows schemas to change quickly as applications evolve. It is provided with the rich querying capabilities. MongoDB is a real time datastore usually used for online data but also find applicability in wide variety of industries.

The MongoDB package has different tools. Depending on operating system, the MongoDB package has different package components. **Mongod**, **mongo** and **mongos** are the core processes of MongoDB package [48]. Mongod is responsible for database whereas mongos is for sharded cluster. Mongo

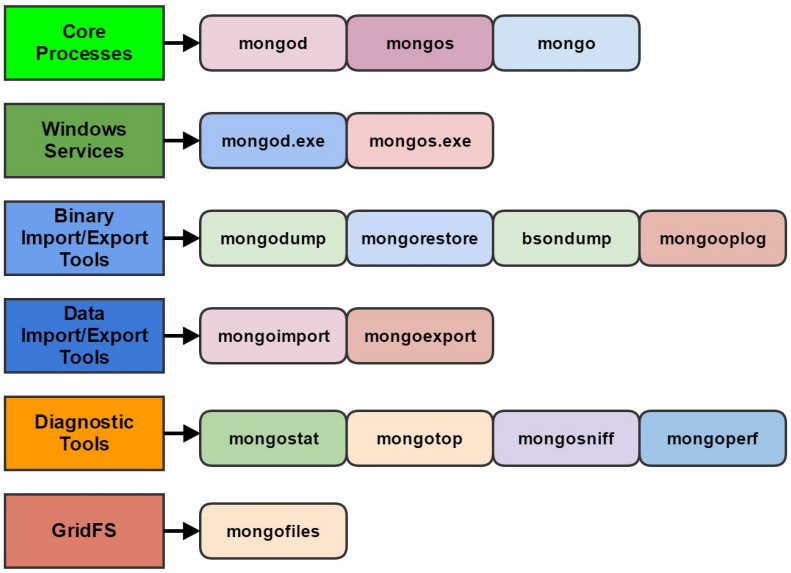

Figure 3.1: MongoDB Package Components

is the interactive shell or the client. For the Windows environment, there are specific services like mongod.exe and mongos.exe. Different tools for data and binary import/export functionalities are the part of MongoDB package. Different MongoDB tools are depicted in the Figure 3.1.

Mongod is the primary daemon process for the MongoDB system. It takes care of data requests, manages data format and executes background management operations. Datastore is a physical container for collections. Each datastore gets its own set of files on the file system. A single MongoDB server typically has multiple datastores. Unlike Extensible Record store datastore like HBase, MongoDB does not require a file system to run.

"Collection" is a group of MongoDB documents [22]. It is equivalent to a RDBMS table. Collections do not enforce any type of schema. Documents within a collection can have different fields. Normally, all documents in a collection are of similar or related purpose. Inside one collection, user can have "n" number of documents. Document has a JavaScript Object Notation (JSON) structure that stores a set of key/value pairs. Normally, all documents in a collection are of similar purpose.

3.1.1.1 MongoDB Development Environment

The MongoDB architecture provides a scalable environment, shown in the Figure 3.2.

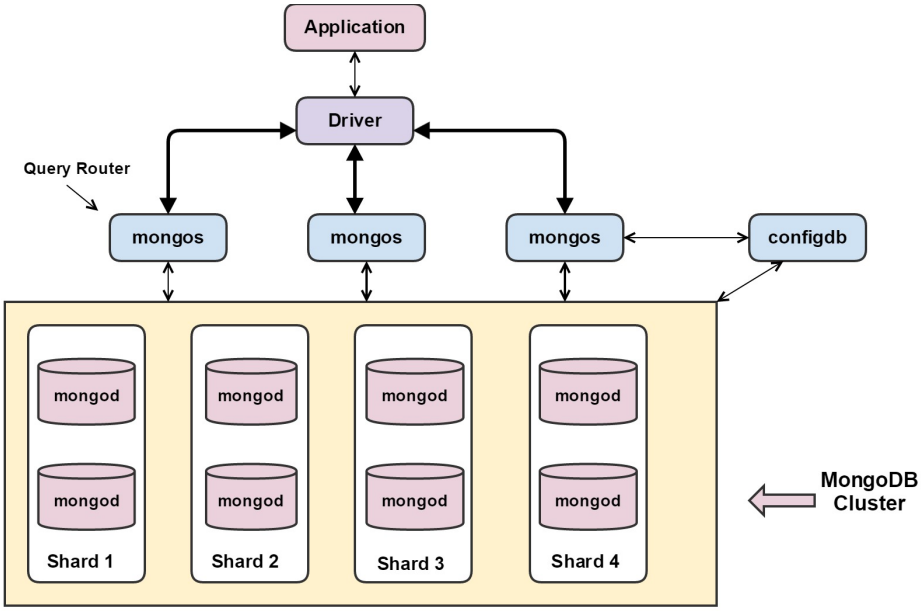

Figure 3.2: MongoDB Architecture

When there is single instance of mongod, the user can connect either using application driver or a mongo client. In case of multiple nodes or distributed cluster, mongos is used as query router that connects mongo client and application driver to mongod shard. To do so, mongos takes help from a configuration database known as configdb. The configdb is an another mongo process that contains metadata of the actual data which is being stored in mongod instances. Single node is called Shard. One Shard contains multiple instances of mongod.

MongoDB has inbuilt capability to horizontally scale out its architecture for low cost community hardware using sharding [64]. Hardware limitations of a single server can be overcome using sharding. The data is automatically balanced in mongod instances and size of the shard increases or decreases as per data size. Three types of sharding techniques are available in MongoDB: hash–based, range–based and location–based.

- Range–Based Sharding: A shard key value is used to partition documents across shards.
- Hash–Based Sharding: MD5 [37] hash of shard key value is used for document distribution.
- Location–Based Sharding: The users specify shard key ranges. They can also provide location base specification for storing the data.

3.1.2 Data Model

MongoDB is a document oriented datastore. Data is stored in these documents in a binary representation known as Binary JSON (BSON). Every document has a unique key, "_id", in a collection [22]. Duplicate keys are not allowed in MongoDB. Keys are strings with some exceptions:

- The null character is not used in the key because null character specifies end of the key.
- The '.' and '$' has special properties and must be used under definite scenario.

MongoDB is type–sensitive and case–sensitive datastore. For example, below two documents are different:

{ "Integer" : 4}
{ "integer" : 4}

Collections in MongoDB have ***Dynamic schemas***. It is possible to store different "shapes" of documents within a single collection. The following documents can be stored in a single collection:

{ "Name" : "Soren"}
{ "integer" : 4}

MongoDB supports four basic operations, create, read, update and delete [22]. User can manipulate data using these operations via shell or mongo client. Let us now briefly look into these operations.

3.1.2.1 Create Operation

A document is added to a collection using insert function. For example, if we want to create a document " thesis", we first create a JavaScript object that will be represented by a local variable "thesis". It will have keys "title", "content" and "submission date".

```
>post = {"title" : "The Survey of NoSQL datastores",
... "content" : "Here is my thesis.",
... "date" : new Date() }
{
"title" : "The Survey of NoSQL datastores",
"content" : "Here is my thesis.",
"date" : ISODate("2014–03–24T20:16:09.982Z")
}
```

db.thesis.insert(post) method is used to insert the above document. The db.thesis.find() method can be used to check if document is saved properly or not.

3.1.2.2 Read Operation

Find and **findone** methods are used to query document in a MongoDB datastore. An empty query gives all the results of a given collection.

db.thesis.find()

{
"_id" : ObjectId("5037ee4a1084eb3ggeef7889"),
"title" : "The Survey of NoSQL datastores",
"content" : "Here is my thesis.",
"date" : ISODate("2014–03–24T20:16:09.982Z")
}

Additional key/value pairs can further restrict the query.

db.thesis.find({"title": "The Survey of NoSQL datastores"})

MongoDB supports a rich query model. For more details, book "MongoDB–The Definitive Guide" can be referred [48].

3.1.2.3 Update and Delete

Similar to read/write operations, update and delete operations are used to modify data.

db.thesis.update({title : "My New Minor Thesis"}, post)
db.blog.remove({title : "The Survey of NoSQL datastores"})

The remove function if not supplied with any key/value pair, removes all documents from the collection.

3.1.3 Query Model

MongoDB has support for different types of queries. The variety of queries is very useful for high scalable operations and analytic applications. Based on parameters, a query can return either a document or a set of specific fields within a document [23].

- Key–Value Query: Any parameter (field) in document is used to query document, often primary key.

- Range Queries: Greater than, less than or any other range is used for querying a document or set of documents.
- Geospacial Query: Returns results subject to proximity criteria, intersection or inclusion as defined by any shape (point, line, circle, square).
- Text Search queries: Text arguments are passed as query parameters.
- Aggregation Framework queries: Aggregation functions like min, max, count etc. are used to query specific results. It is similar to GROUP BY function of SQL.
- MapReduce queries: It is specifically defined for data processing specified in JavaScript.

3.1.4 Properties

MongoDB is a powerful and flexible Document Oriented datastore. Flexible data model coupled with dynamic schema provide base for fast iterative development process. MongoDB is a scalable datastore and it is easily scalable within and across different data centers. MongoDB has a pluggable storage architecture. In its latest release, MongoDB 3.0 supports Memory Mapped Version Engine (a type of storage engine) and has a future scope for integration with HDFS. MongoDB has rich secondary indexes including geospacial and TTL indexes [23]. It has build in replication for high availability. ACID properties are supported at document level. The excessive use of RAM increases the speed of database operations in MongoDB.

3.1.5 Fault Tolerance

MongoDB provides native replication and automated failover which gives high reliability and operational flexibility. Replication provides redundancy and increase data availability. Replica set is a group of mongod instances that host the same data set. It can be also viewed as a cluster of N nodes. The primary mongod instance receive all write operations. The replica set can only have one primary instance though any node can become primary through leader election in case of failure. When the client sends data to primary replica, it gets replicated to secondary replica nodes. One another node, Arbiter is also part of Replica set but it does not hold any data. Arbiter triggers selections and vote in selecting the primary node. Heartbeat is circulated among the nodes to check their alive status. If Arbiter does not

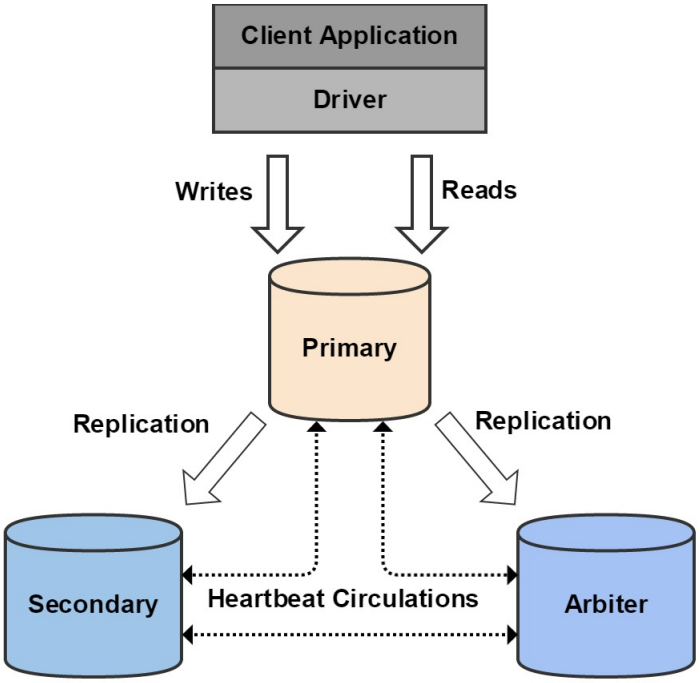

Figure 3.3: Replication in MongoDB

receive signals from primary nodes for a given time, it triggers leader election to select another primary node.

It is also important to note that the MongoDB is a CP datastore but it has few bugs [1] [2]. A recent Aphyr's blog post [13] shows that MongoDB's consistency model is broken by design. It is also observed by them that MongoDB sometimes read stale version of documents and can also show garbage data from non occurring writes.

[1] MongoDB bug: https://jira.mongodb.org/browse/SERVER-17975 (accessed: 11/05/2015)

[2] MongoDB bug: https://jira.mongodb.org/browse/SERVER-18022 (accessed: 11/05/2015)

Chapter 4

Extensible Record Datastores

Google's Big Table is the motivation for extensible record database engines. It has a flexible data model with rows and columns which can be extended any time. Apache HBase, Apache Accumulo, and HyperTable are few of the famous Extensible Record stores. Extensible record stores are scalable and both rows and columns can split over multiple nodes. Extensible Record stores are often term as Column Oriented data stores.

Apache HBase is the popular open source Extensible Record datastore. We will discuss its detail in following section.

4.1 HBase

HBase is a Column Oriented data store that runs on top of HDFS. HBase is an open source Apache project which can be summarized as distributed, fault tolerant scalable data store. It is good in managing sparse data sets.

Unlike a relational database management system (RDBMS), it does not support structured query language like SQL. In fact, HBase is not at all a relational database. HBase is written in Java much like a typical Hadoop application but it does not use MapReduce. HBase applications can also be written using AVRO, REST and THRIFT API. A HBase system is made up of set of tables. These tables are stored in HDFS. Each table contains rows and columns much like a traditional database. Each table has a column defines as it primary key and all calls to access the table must use the primary key.

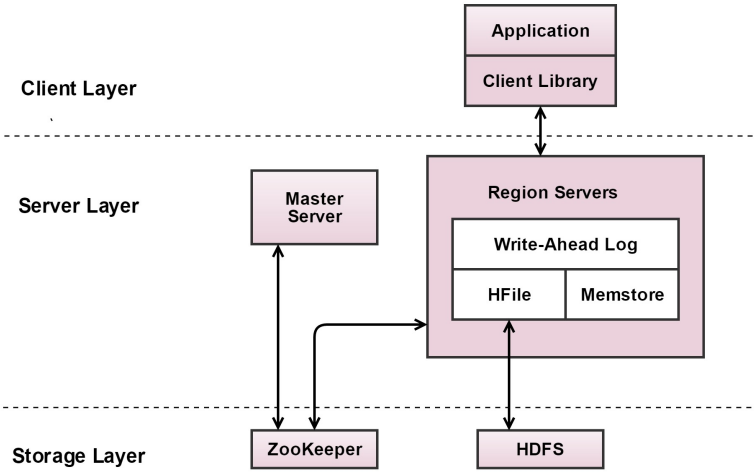

Figure 4.1: HBase Architecture

4.1.1 Architecture

HBase architecture has three layers namely: the client layer, the server layer and the storage layer. The client layer provides an interface to the user. It has client library which is used to communicate with the HBase installation. The storage layer has a coordination system and a file system. HDFS is the most commonly used file system for HBase. Apache ZooKeeper [66] [10] is used as the coordination service for HBase. A master server and the region servers are two component of server layer. The Figure 4.1 gives an architecture overview of HBase.

4.1.1.1 The Storage Layer

The storage layer has two components:

- File System

  HBase mostly uses HDFS as its file system. HDFS easily handles large datasets and provides streamline access to data. It provides a scalable and robust mechanism to HBase for the storage of files. HDFS can very well handle data replication between nodes, therefore even if a storage server is offline, it would not affect the data availability.

- Coordination Service

  Apache Software Foundation has released an open source service called Apache Zookeeper. Apache ZooKeeper [10] extracts its design con-

cept from Google Chubby which is used for BigTable. ZooKeeper is a highly reliable distributed coordination service that provides an interface similar to Filesystem, where each node (znode) may contain data and children. In HBase, Apache ZooKeeper is used for coordination and communication of states between master server and region servers. A personal node is created in ZooKeeper by every region server which is used by master to look for available servers. Server failure and network partitions are also tracked by these ephemeral nodes. ZooKeeper ensures only one master runs at a time in HBase and used for storing bootstrap location for region discovery [60].

4.1.1.2 The Server Layer

The server layer consists of two parts:

- Master Server

  HBase architecture is based on master–slave model in which master is responsible to take decisions and one or more slaves work together to complete the given job. Master server is implemented as HMaster [60]. It has an external interface which interacts with the region servers, client and other utilities. In a cluster, master monitors all the region servers and takes care of all metadata changes. It also handles load balancing across region servers.

  Master usually runs on NameNode. Master server is lightly loaded because it is not involved in data storage or data retrieval process. Rather, it maintains the current state of distributed cluster. A client talks to master server via ZooKeeper so even if master server goes down, a distributed cluster can still run further. But it is recommended to start the master as soon as possible.

- Region Server

  In HBase, table is the logical view of data storage and adjacent ranges of rows are stored together in a region which is a physical layout [60]. A region is the primary unit of scalability and load balancing. Initially, one region is assigned per table and data get stored in this region until the maximum configured size of the region. If this limit if exceeded, the region further gets split into two at the row key in the middle of the region. A row key is the unique identifier for every row in the table. The splitting process in very instantaneous because the system creates two reference files for newly splitted regions. Parent region does not

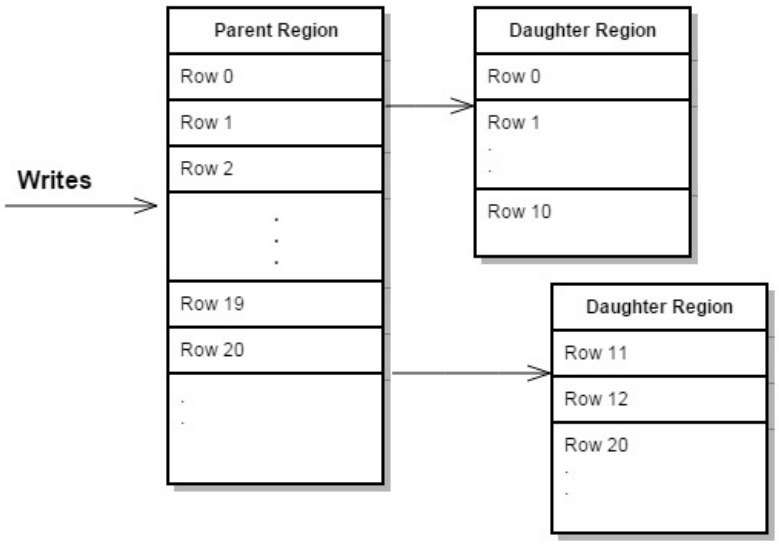

Figure 4.2: HBase Region Splits

receive any further request after the splitting. The splitted regions are also known as daughter regions [60].

A region server is implemented as HRegionServer [60]. It manages and serves the set of regions. One region can only be served by a single region server. In a distributed cluster, region servers run on DataNodes. The ".META." table is a system table responsible for storing the mapping of regions and region servers. So client can bypass the master and directly communicate with region server to retrieve data from a particular region.

4.1.1.3 The Client Layer

The HBase client is responsible to find the region server serving a particular range of rows. The client does it by querying the ".META." table. For this purpose, the client communicates with the ZooKeeper server to get the information about the "–ROOT –" table. The "–ROOT –" table provides reference to all the regions in ".META." tables. In this step, the client retrieves the name of the region server hosting that specific region. Now the client looks for matching ".META." region in the "–ROOT –" table. Finally

the client reads the ".META." table to retrieve the correct user table region. This three level look-up process exclude any role of master server in in storing or retrieving of data. The client optimized the complete process by caching the look-up information so that next time client can directly access the user table region. If region server dies due to any reason, the client then again performs these three steps to determine the new location of the user table region.

4.1.2 Data Model

The initial design of HBase data model is derived from Google Big Table. The data model is exclusively designed for semi structured data that may differ in columns, data type and field size. Its layout has a flexible design which enables an easier partitioning and distribution of data across cluster. The data items are stored in key/value pairs. Both keys and values are a byte array that means user can store anything in them. For example, Facebook uses HBase to store its chat and messages [38].

Each data value is identified with a unique key. The keys are multi-dimensional and they are indexed by a row key, a column key and a timestamp. A key is commonly represented as the tuple which points to a specific value:

(row key, column, timestamp) −>value

Now, let us briefly discuss some of the logical components of data model.

1. Table

   The data is organized into tables. Strings are used to define table names. Table names are made up of set of characters that are safe for defining the file system path.

2. Row

   Column is the basic unit of the data model. One or more column together make a row. Within a table, rows are used to store implicit data items. Each row has a unique identification known as row key. The row key is a byte array.

3. Column Families

   Rows are grouped into column families. Column families are the main unit of semantic boundaries between data stored in rows. Column families are defined up front while creating a table and it is not recommend to change them quite often.

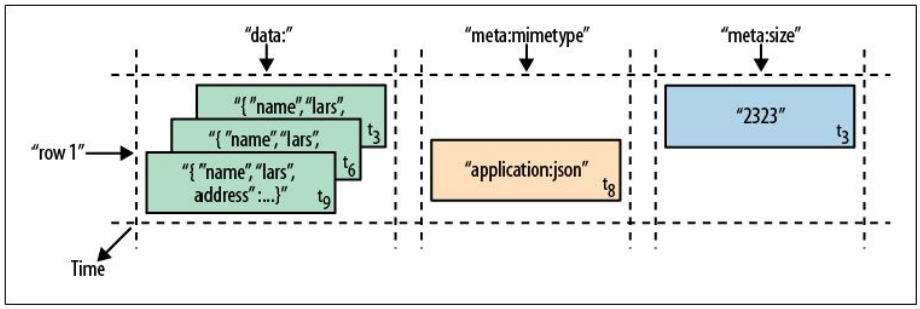

Figure 4.3: A Time Oriented View of a Row Extracted from [60]

4. Column Qualifier

   The *family:qualifier* is used to reference a columns where *qualifier* can be any arbitrary array of bytes. They do not have any data type like a row key.

5. Cell

   The combination of row key, column family and column qualifier is called cell. The data is stored in cell as its value and cells are grouped as rows.

6. Timestamp

   For each cell, timestamps are implicitly defined by system or explicitly set by users. Timestamps are used for the versioning of the data. A cell can have multiple versions. Internally it is used for concurrency control. The versions of the cell are sorted in descending order so user can see the latest version first. The number of cell versions is set to three by default.

4.1.3 Data Storage

HBase storage design is based on Log–Structured Merge–Trees (LSM Trees) [79]. In HBase Trees, the incoming data is firstly stored in a log file. Once the modified log file is saved, an in–memory store is updated with the recent modification of log files. This in–memory store is used for fast lookup. When the in–memory store is full with updates, a new store file is created in the disc and all updates from in–memory store is flushed to it.

The HFile class implements the actual storage files in HBase. These files are known as HFiles/ Store Files. HFiles are the replica of the SSTable format

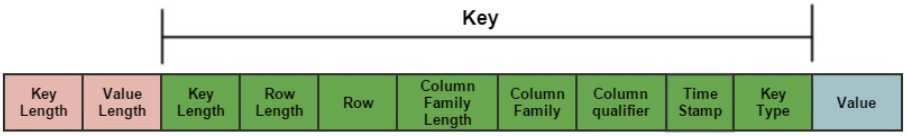

Figure 4.4: The Key Value Format

used in Google Bigtable architecture design. Earlier, the key–value instances are stored inside HFiles in a key value format as depicted in the Figure 4.4 [60]. But in HBase 0.92 and in later versions, HFiles are changed to HFile v2 for better speed, performance and cache usage [8]. The key consists of the row key, the column family name, the column qualifier and so on. For a small payload, it is recommended to keep the key short otherwise it creates a large overhead.

4.1.3.1 Write

In HBase, the row is initially written to a commit log. The client issues a request to the Region Server which in return handover the details of matching region. HBase stores the row in Write Ahead Log (WAL) [60]. The Write Ahead log contains HLogKey instances. The keys stored in HLogKey consists of a sequence number and the actual data and these keys are used to recover not–yet–persisted data in case of server failure. WAL stores all the modification done in a particular region and provides atomicity and durability. Then the rows are written to in–memory memstore. The data in the memstore is ordered. Once the memstore is full, the data is flushed to HDFS disc. The actual data is stored in HFile.

There are two ways to insert data into HBase. The first one is very simple way where the user have little amount of data. Therefore, user can write a single column into HBase. For this, client can directly connect to HMaster. HMaster further assigns a region server and a region to write. When memstore is full, the data is persisted to HDFS storage. When user have several terabytes of data, it is not good to write directly to HMaster otherwise it will be overwhelmed with excessive data. In this case, user can utilize bulk write facility of HBase where user can directly write to HFiles by accessing corresponding regions. In this process, the client finds the address of the region server in ZooKeeper. The client reads the address and then writes row by accessing the region server.

Each region might contain many HFiles because many HFiles are created when data is flushed to disk. Higher number of HFile results into more seeks and therefore higher latency during reads. For better performance, HBase performs automatic merging of multiple HFiles and this process is known as compaction. There are two types of compaction [60]:

1. Major compaction

   In major compaction, all files are compact into a single file. User can force the major compaction through command line or server can automatically checks for major compaction if its due. The server parameters for the same are implicitly defined (*base.hregion.majorcompaction*=24 hours)

2. Minor compaction

   It is responsible for compacting last few files into a single large file. When major compaction is not due, minor compaction is automatically assumed by the server.

4.1.3.2 Indexes in HBase

As we have discussed earlier, all the rows are always ordered by row key in HBase. Each key/value pair has multiple parts such as row key, column family, column, version, and value. HBase does not have native support for indexes, instead a table can be created that serve the same purpose. HBase provides support for bloom filters that is used to find out quickly if particular row/column combination exists or not.

4.1.3.3 Read

This section explains about the read operation in HBase. When the client want a specific row and place a get request to the Region Server, it checks firstly in the memstore. Before scanning all the HFiles, a quick exclusion check is performed using timestamp and bloom filter to exclude rows those are definitely not related to requested value. Therefore, when the desired row is not present in memstore, HFiles are searched from newest to oldest stored data.

4.1.3.4 Delete

It is important to note that rows are never directly deleted from HFiles. When client place a delete request, the Region Server search for the desired row and points a delete marker to it. In case if a Get request tries to access

the same row, the data will not be returned because of the delete marker. In the next scheduled major compaction, the specific row is finally deleted.

4.1.4 The Client API

The HBase and its native API are essentially written in Java. Java basically enables programmatic capabilities to interact with HBase. As noted earlier, there are other API that are available to communicate with HBase. HBase applications are deployed as JAR files on top of Hadoop cluster. The user build entire HBase application and run it as a MapReduce job. It is important to note that all the operations to change data in a single row are atomic [46]. HBase provides row level locking and transactions [87]. Any number of columns can be covered for a single row and atomicity of the operation is guaranteed. Hence it directly affect other concurrent users. The concurrent users either can read the last modified changes or they have to wait till release of row level lock to apply their changes.

HTable class is the client interface to HBase. It enables user to get and put data in HBase as well as delete the data when required. Since HBase version 0.99.0, HTable is not acknowledged as client API and replaced with the interface "Table". Now we will discuss some of the client API operations in HBase.

4.1.4.1 Put Operation

Put command allows users to insert data into HBase. The same command can be used to update the data. Put can be grouped into two separate operations, one is for single low and second is for list of rows.

Void put(Put put) throw IO exception is the user call that creates one or lists of put objects using following constructors:

Put(byte[] row)
Put(byte[] row, RowLock rowLock)
Put(byte[] row, long ts)
Put(byte[] row, long ts, RowLock rowLock)

User provide a row identified with row key and then can add values to it using add() methods:

Put add(byte[] family, byte[] qualifier, byte[] value)
Put add(byte[] family, byte[] qualifier, long ts, byte[] value)

Besides row key, user can explicitly provide version number for a given key instance. In case if its not mentioned, HBase gives an automatic timestamp (version number) created from the current time of region server accountable for the particular row.

We have discussed earlier that a table can have multiple column families and a column family consists of group of columns. If a user want to insert a value "value1" in table "SampleTable", the sample shell command for this operation is:

put 'SampleTable', 'rowkey1', 'cf:cq', 'value'

where *cf* and *cq* are column family and column qualifier respectively.

Void put(List <Put>puts) throws IOException is the call which creates list of Put instances. Whether its a Single Put or Lists of Puts, they are remote procedure calls (RPC). To reduce the overhead, HBase client API has *write buffer* which combines Put operation and send them in a single RPC call.

Put calls have another variation where user can combine Put and Check calls as:

checkAndPut(byte[] row, byte[] family, byte[] qualifier,byte[] value, Put put)

In this method Puts are checked with checking points. Only if checking points are completed, the Put operation is executed. This operation is invalid for different keys and used exclusively for a single row.

4.1.4.2 Get Operation

The Get command retrieves data from HBase table. Like Put operation, it also has two groups for single and multiple rows separately. The get operation is also atomic in nature and bound to single row but it can retrieve multiple columns. For example, a basic shell command for get operation is:

get 'SampleTable', 'rowkey'

The above example will retrieve data for all columns within all column families for that particular row with a specific key. User provides row key in order to get key/value pairs.

Result get(Get get) throws IOException is the method that retrieves specific values. User provides Get object that is created by one of following constructors:

Get(byte[] row)
Get(byte[] row, RowLock rowLock)

User can add more parameters for more specific search. For example filters can be applied to get a column family, column qualifier or to retrieve given number of versions. Some of these filters are:

Get addFamily(byte[] family)
Get addColumn(byte[] family, byte[] qualifier)
Get setTimeRange(long minStamp, long maxStamp) throws IOException
Get setMaxVersions()
Get setMaxVersions(int maxVersions) throws IOException

The Result class returned from get() method contains all the matching results. If there is a need to further limit the results, here are some methods user can use:

byte[] getValue(byte[] family, byte[] qualifier)
byte[] value()
byte[] getRow()

The getValue() retrieves data for a specific cell in HBase. The value() call returns the data from the most recent cell in the first column found. These methods are very good in flitering out unwanted data.

4.1.4.3 Delete Operation

Delete command delete data from HBase table. To delete a cell from "SampleTable" at row key "rowkey1" marked with the timestamp "t1", a shell command needs to include all parameters:

delete 'SampleTable', 'rowkey1', 'cf:cq', 'ts1'

User can delete single row or lists of rows. *Void delete(Delete delete) throws IOException* creates delete instance and addition to it, following constructors create delete objects:

Delete(byte[] row)
Delete(byte[] row, long timestamp, RowLock rowLock)

It is also possible with additional delete methods to narrow down delete preferences. Method like *Delete deleteFamily(byte[] family)* removes entire column family.

4.1.4.4 Scan Operation

Scan operations are used to iterate on HBase tables to perform some operations over a range of data. These operations can by anything like counting number of occurrence of a name for a given time period. Instead of having similar function call like get(), scan has getScanner() which returns scanner instance user want to iterate. The different available methods are:

ResultScanner getScanner(Scan scan) throws IOException
ResultScanner getScanner(byte[] family) throws IOException
ResultScanner getScanner(byte[] family, byte[] qualifier) throws IOException

Here getScanner method returns ResultScanner class. The ResultScanner converts the Result instance for each row in the range into an iterator functionality.Just like Get and Put operations, Scan also creates different constructors:

Scan() Scan(byte[] startRow, Filter filter)
Scan(byte[] startRow)
Scan(byte[] startRow, byte[] stopRow)

In Scan operation, it is mandatory to specify the startRow parameter while defining stopRow is exclusive.There are different filters to narrow down scan such as *Scan.addFamily()* and *Scan.addColumn()*.

All the above mentioned operations are explained illustratively in the book "HBase: The Definitive Guide" [60].

4.1.5 Properties

In this section we discuss some of the properties of HBase.

HBase is a schema less, wide and denormalized Column–Oriented datastore on top of HDFS/Hadoop. HBase is very fast for aggregation because query needs to hit just the array of values stored in a column. It also provides high rates for compression.

HBase is distributed and highly scalable over multiple nodes. Automatic sharding on primary key is an important feature of HBase. HBase is a CP system since it supports consistency and partition tolerance properties according to CAP Theorem. Stronger level of consistency is supported by giving up some availability. After any write, the next read sees the latest version of the value stored in HBase because at a given time, only one region server manages the key associated with that value. Availability is sacrificed

because if a Region Server fails, its data is unavailable until another Region Server replaces it.

HBase does not support multi–row ACID transactions like RDBMS but individual row operations are atomic. These operations are supported by row level locking and transactions. It has a write optimized design and read performance is sacrificed. These writes are sequential.

4.1.6 Fault Tolerance

Fault Tolerance is a key aspect of HBase design. HBase easily survives the message loss because of network problems and server failures. If a Region Server dies, the master server designates a new Region Server for each region the collapsed server served. And if the master fails, the backup master will take over its place. The client search for master server and region servers via ZooKeeper which itself is fault tolerant. ZooKeeper also has role in failure detection. It is used to identify failed Region Servers. It can be possible that ZooKeeper might declare a region dead when it is not, so master needs to establish connection with that region before recovery starts to confirm the failure.

HBase itself does not replicate data. The underlying distributed file system like HDFS supplies replication. This means HBase needs a distributed file system for its fault tolerance abilities.

Chapter 5

Key-Value Datastores

Key–value datastores are primarily a big hash tables with unique primary key and a pointer to a particular data item. Its data model has identical design to the memcached in memory cache. The keys can be primitive types or objects and values are accessed only by keys. These data stores provide support for many functionalities like replication, partition, locking, versioning, transactions and/or other features. They are extremely useful in building specialised application with super fast query capabilities. Cassandra, Redis, Riak, Scalaris, and Project Voldemort are few examples of key–value datastores. We discuss some of them briefly in this section.

5.1 Cassandra

Cassandra [70] is a distributed, highly scalable and fault tolerant NoSQL datastore. It is a structured store with decentralized architecture. It was developed by Facebook Inc. and its first release came out in 2008. The main aim to develop Cassandra is to meet storage requirements of the Index Search Problem [70]. For this purpose, Facebook needed a datastore with very high write throughput. Apache Cassandra [5] is an open source project under the Apache license 2.0. This section summarises architecture design and properties of Cassandra.

5.1.1 Data Model

The data in Cassandra is stored in distributed multi dimensional tables indexed by a key. The value is structured object. The tables have unique row key for ever row with no size limit. Typically their size ranges from 16 to 36 byte. Cluster is very similar to database server which has many instances

and each instance is responsible for set of databases.

Cassandra has a ring shaped cluster with multiple nodes. Cluster is a container for key spaces. Key space is basically a database if we compare Cassandra design with RDBMS. The key spaces are used to group column families together. Column family is a set of columns. Column family inherits its design from Google's Bigtable [47]. Mostly, a cluster has one key space per application. Key space has a name and three attributes. These attributes are:

- Replication Factor

  The number of copies of each row of data stored in the cluster is known as replication factor. For example, if we set replication factor as five, then five nodes will store copy of rows. Replication factor is an important parameter to achieve high consistency level and it is transparent to the client.

- Replica Placement Strategy

  Replica placement strategy defines how the replicas are actually placed in the cluster. There are three different strategies that determine which replicas will occupy which nodes. These strategies are Simple Strategy, Old Network Topology Strategy, and Network Topology Strategy [63].

- Column Family

  A table in a RDBMS is analogous to column family. Column family is the container for the collection of rows. Each row comprises ordered columns. The structure of stored data is represented by column family. Cassandra exclusively creates column family to be a logical division to associate similar data. The Column family has two attributes, a name, and a comparator.

The column is the smallest increment of data in Cassandra. A Cassandra column family can have either simple columns or super columns. Super columns add another level of nesting to the conventional column family structure [69]. The super column contains a super column name and an ordered map of sub columns.

Every column in Cassandra has three attributes: a name, a value, and a timestamp. The name, the value, and the key can be of different data types. The data type for a column value is called validator whereas data type for column name is known as comparator.

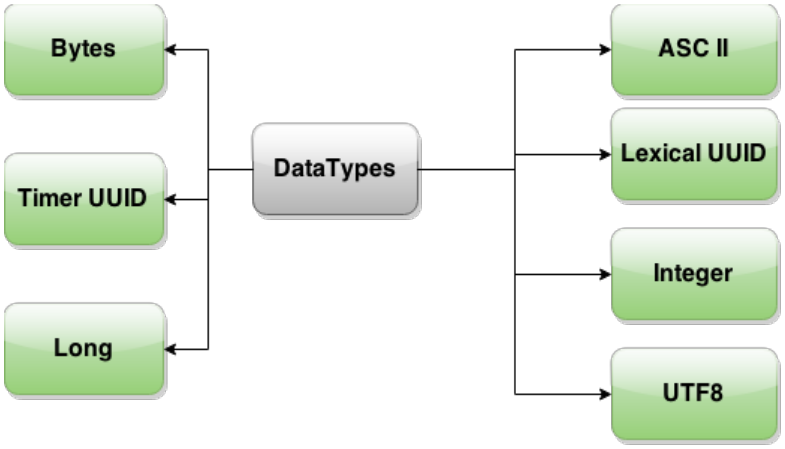

Figure 5.1: Different Data Types Supported by Cassandra

5.1.2 Architecture

In traditional databases that can be deployed over multiple nodes and even in datastores like HBase, Google's Bigtable etc, master slave relationship exist between the nodes. The master is authority for distributing and managing data. Slaves on other hand synchronize their data to the master. All writes pass via master and it is the single point of failure. The architectures that have master/slave setup sometime have adverse effect if master node fails.

By contrast, Cassandra was designed with the understanding that failures can and do occur. It has a peer–to–peer [83] distribution model. The data is divided among all nodes in the cluster. All nodes are structurally identical. Therefore, there is no master node. Equality among nodes due to peer–to–peer network improves general datastore ability. It also makes scaling up and scaling down much easier because a new node will not be treated differently.

There are several different theoretical constructs which supports Cassandra architecture. Le us now discuss some important principles.

5.1.2.1 System Key Space

Cassandra maintains an internal key space that is used to store meta data about the cluster to assist in all kinds of operations [63]. This system key space has information about local node and hinted handoff. We will discuss hinted handoff later in this section. The system meta data comprises of five attributes [63]:

1. The node's token.

2. The cluster name.
3. The key space and the schema definition.
4. The information about migration data.
5. The bootstrapping knowledge related to all the nodes.

5.1.2.2 Commit Logs, Memtables and SSTables

Commit logs are used to support Cassandra's durability goals. It is a crash recovery mechanism. When user gives a write request to Cassandra cluster, the writes are appended to commit logs. There are no updates at this point. Once the data is written to commit logs, Cassandra writes are deemed to be successful. After this, data is written to *memtable*. Memtables are the memory resident data structure. Storing objects into memtable is also an append–only operation. When the memtable is full, its content is flushed to permanent storage disc in a file known as SSTable. Special flags are set up for each memtable that indicates whether it needs a flushing.

In Cassandra, each data item is replicated at N hosts where N is defined as the **replication factor** configured for every instance. The replication helps Cassandra to achieve high availability and durability. The consistency among the replicas are maintained by a quorum like mechanism. Cassandra has tunable consistency and different levels of consistency can be set. These levels are ZERO, ONE, ANY, QUORUM, and ALL [63]. If read consistency level is chosen as QUORUM, the query will reach to all nodes. If majority of replicas respond, the value with the most recent timestamp is returned to the client. If QUORUM is set for write consistency level, the write is received by at least majority of replicas. Cassandra can keep the replicas consistent with each other in scenarios where there are no server failures.

5.1.2.3 Hinted Handoff

Hinted handoff [57] is a special feature of Cassandra architecture design. It ensures general availability of the ring. In case a node goes down, the other nodes will get the data that belongs to that particular node. The substitute nodes maintain a hint that once the original node joins the ring again, all the data belong to the original node will be transferred back to it.

5.1.2.4 Compaction

Compaction is the process of releasing the occupied space upto an extent by merging large data files. Cassandra performs compaction to merge SSTables.

During compaction, the keys are merged, columns are united and tombstones are discarded [63]. Tombstones are analogous to *soft deletes* in relational databases. Once the process in completed, new indexes are created over freshly merged data. Read only compaction and Major compaction are the two types of compaction supported by Cassandra. The level of compaction can be manually set by the user and it can be done by increasing or decreasing value of the flag *–Dcassandra.compaction.priority=1* [63].

5.1.2.5 Bloom Filter and Anti Entropy

Bloom filter [42] is a space efficient probabilistic data structure used as a performance booster. Burton Bloom invented bloom filter in 1970. Bloom filter checks if an element is a member of a set using fast non–deterministic algorithms. These filters are part of memory and act as special cache that allow quick search.

Anti entropy is the replica synchronization procedure for confirming that data on different nodes is modified to latest version. Cassandra nodes exchange *Merkle trees* [74] for conversation with neighbours during a major compaction. While reading data, client connects to any node in the cluster. The client sets the consistency level and based on it, a number of nodes are read. It might happen that some nodes sent out of date value. To prevent the false read, Cassandra runs read repair process in the background. Read repair helps replicas to get updated with latest values. The Figure 5.2 depicts the read repair process.

5.1.2.6 SEDA

Staged event–driven architecture (SEDA) [94] meant to support massive concurrency demands. Cassandra follows SEDA for its concurrency model. In this model, different operations are divided into stages and some events are associated with every stage. Also, there is different thread pools associated with each stage. SEDA stages are composed of three components: event queue, an event handler, and an associated thread pool [63].

5.1.3 Properties

Cassandra is a highly available open source distributed datastore. It is a key value store where one key can map to one or more values. Unlike relational management databases, Cassandra does not require matching columns within a row. Cassandra determines the correct version by resolving the timestamp. Its decentralised architecture avoids single point of failure and any node can

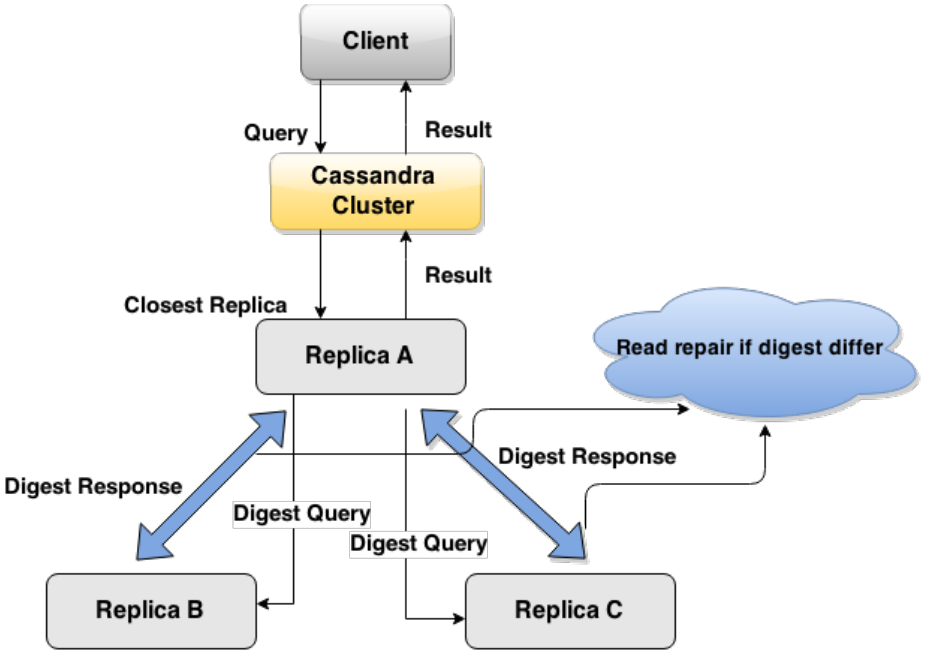

Figure 5.2: Cassandra Read Repair

serve the request. Cassandra supports replication. Replication strategies are configurable based on developer's need.

The nodes communicate between each other using gossip protocol [59], which exchanges information across the cluster every second. Cassandra has tunable consistency. Its consistency can be balanced with the required availability by setting up the consistency levels. Based on the CAP theorem, Cassandra is an AP datastore. It supports eventual consistency. Cassandra comes with drivers for major languages. It also has integration point support for other software like Apache Hive [9], Apache Pig [34], and MapReduce.

5.1.4 Fault Tolerance

Cassandra has ring shape peer–to–peer architecture. All nodes are the same. Cassandra distributes data across all nodes using consistent hashing techniques and data is evenly distributed among the nodes. The data is replicated across N nodes in the cluster. N is the replication factor. A leader is elected and that leader distribute keys among N-1 nodes. Leader is elected using the system known as ZooKeeper [66].

When a node fails, the other nodes acquire the data belongs to that particular node. The substitute nodes maintain a hint. Once the original node joins the ring again, all the data belong to the original node will be

transferred back to it. If failed node is the leader, then again a new leader is elected and ring starts to function in normal way.

5.2 Riak

Riak is a NoSQL datastore for scalability, distribution and fault tolerance. It is a highly available open source key-value datastore. Riak is considered as a clone of Apache Dynamo [82]. It is written mostly in Erlang with some C/C++.

Basho [11] is a company founded in 2007 by ex Akamai employees. Basho sponsors Riak open source and sells Riak enterprise solution. Let us now discuss some of the aspects of Riak design in next few subsections.

5.2.1 Riak Components

Riak is a key–value datastore that distributes data over multiple physical servers. These physical servers are known as nodes. These nodes collectively form a cluster. Riak cluster is in ring structure.

Riak can be assumed as a very large hash space. The value is hashed into specific location using a unique hashing key and it can be retrieve anytime.

5.2.1.1 Object

Objects are the basic unit of data storage. Riak object consists of a key, a value, a bucket, a clock and associated meta data [82]. The programmers define keys. We can assume key like a home address, for instance a unique key 1989 represents Bob's home address in Riak hash space. Therefore, we can retrieve Bob using this unique key.

$$\text{hashtable}[\text{"1989"}] = \text{"Bob"}$$
$$\text{"Bob"} = \text{hashtable}[\text{"1989"}]$$

The key is unique but it does not mean that it can be only used once. Programmer can store another value with the same key if the previous value is no longer in use.

5.2.1.2 Buckets

Riak objects are stored in JSON format and have many fields. Buckets are the collection of objects. Similar to *collections* in MongoDB, buckets provide logical namespace to prevent the collision of identical keys in different buckets. There are number of configuration properties for each bucket:

- n_val: It defines the total number of copies each object that are stored in Riak cluster. By default, its value is three.
- allow_mult: It takes boolean values and determines if siblings can be created. We will discuss siblings later in this section.
- last_write_wins: This parameter is used to resolve update conflicts on Riak objects. Its value is either true or false.
- r, pr, w, dw, pw, rw: These parameters define the number of responses needed before an operation is accepted successfully.
- precommit and postcommit: The functions need to be executed before and after creating an object.

5.2.1.3 Vector Clocks

In a distributed system, vector clocks help to determine how different events are related through their timestamps. In 1978, Lamport had defined the "Happened Before " relation on set of events in a system. Vector clocks are based on causal ordering of events [72]. Causal dependency or independency of events depends on "Happened Before" relation [72].

Vector clock establishes an initial version of each value stored in Riak. With every update of a value, a new timestamp is stored in vector clock for the updated value. Different versions of the object is compared to determine:

- If specific objects have direct descendant relation or not.
- If objects have common parent.
- If compared objects are unrelated.

5.2.1.4 Siblings

On many occasions, it is not possible for Riak to differentiate between canonical versions of a stored object. At this moment, siblings are created under three circumstances:

1. Concurrent Writes

   When the clients having same vector clock value simultaneously try two writes, in this situation Riak can not figure out the correct object to store and therefore, two siblings are created for the object.

2. Stale Vector Clock

   When causally dependent read/write request from different clients use stale vector clock value in either operation, random siblings are created by the client which previously issued writes using stale clock values.

3. Missing Vector Clock

   When a client issues a write to an object without a vector clock value, siblings are created.

5.2.2 Riak Operations

Riak is a key/value datastore, so most of the interactions are setting or getting values. It supports many interfaces using HTTP, Erlang, Java, Python etc.

Mainly, Riak operations are to get,to put,to post or to delete a value. If we want to retrieve specific key from the bucket, following command can be used:

GET /riak/bucket/key

The basic request to store a value in a bucket, PUT operation is used.

PUT /riak/bucket/key

For example, if we want to put the value *Kabab* into the key *cuisine* under the *indian* bucket and *items* bucket types, it can be done in this way using HTTP method in curl:

curl –XPUT "$RIAK/types/items/buckets/indian/keys/cuisine \
–H Content–Type: text/plain" \
–d "Kabab"

To delete an object, pattern like GET and POST is followed:

DELETE /riak/bucket/key

Indexing is very useful for document search in Riak. All schema definitions for indexes are stored in the bucket " _rs_schema". For example, the schema for the *cuisine* index is saved under "_rs_schema\cuisine".

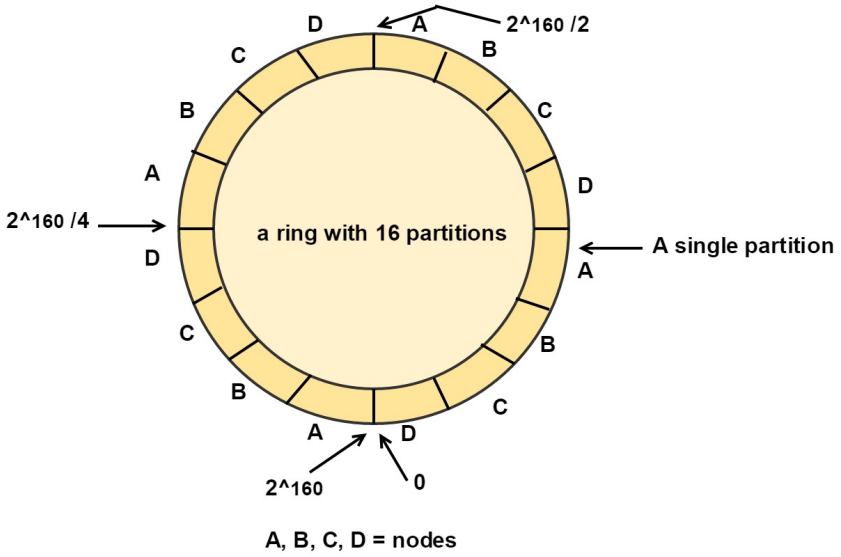

Figure 5.3: Riak Ring

5.2.3 Architecture

Riak maps object into a ring shaped cluster using consistent hashing [68] techniques. The maximum hash value for any Riak cluster is 2^{160} and it is divided into equal sized partitions. Every node in the cluster is a separate physical server accompanied with few virtual nodes, or "vnodes" [82]. Each vnode declares a partition on the ring. The total number of vnodes in a Riak ring are calculated by (number of partitions)/ (number of nodes).

The number of partitions by default is 64. Therefore, in a 64 node cluster, each node is responsible for 1/64 of the ring. A partition is a way to divide a set of keys on separate nodes (physical servers). One server hosts a range of keys and other servers are responsible for remaining non-overlapping ranges. For example, if we want to partition a set of 100 numbers into 2 servers, Riak might put first 50 numbers into Node A, and rest into Node B.

5.2.4 Data Types

In the version 2.0 or later, Riak specific data types are included. These data types are motivated from Convergent or Commutative Replicated Data Types (CRDTs) [86]. Riak has five CRDT inspired data types: counters, flags, maps, sets and registers. State based replication and operation based replication are the two flavors of CRDT [85].

To use these data types, it is necessary to create a bucket type. Bucket

type sets the value of datatype parameter. This value may take counter, map or set as input. At the point of data type conflicts, few conflict resolution rules are applied specific to a particular data type. The convergence rules for five data types of Riak are given below.

1. Flags: Enable get priority over disable.
2. Registers: The most recent value based on timestamp wins.
3. Sets: When an element is simultaneously added or removed, add wins over remove.
4. Maps: Add/update win when a field is simultaneously added, removed or updated.
5. Counter: A separate count for increment and decrement is maintained for each actor. At the time of merge, maximum value of the count will win. For example, value of an actor is 19 and value of other actor is 20, Riak will take 20 as the final value.

5.2.5 Fault Tolerance

Riak has an important property that it is master less. This means every node in Riak is same and have independent copy of Riak package. Lack of master-slave relationship among the nodes provides a uniformity in Riak architecture. This uniformity is the basis for fault tolerant architecture of Riak.

In Riak, the data is evenly distributed across nodes. For this purpose, consistent hashing [68] is used. The data is also replicated in the cluster and by default there are three replicas per object. Even if a node fails, the data is still available. The neighbouring node takes care of all storage operations of failed node. When the failed node recovers and restores network connectivity, neighbouring node handovers all the updates to the failed node that are saved on neighbouring node in the absence of original node. This technique is known as Hinted Handoff [57].

5.2.6 Properties

Riak is a distributed datastore platform designed to solve problems associated with building and scaling read and write intensive applications. It is fault tolerant, highly available datastore. Riak is capable to store any type of data across multiple nodes in the cluster. In Riak cluster, data is replicated for

partition tolerance. Its horizontal scaling is supported by consistent hashing and vnodes.

Eric Brewer had explained trade off between availability, consistency and partition tolerance in CAP theorem for distributed databases [43]. Riak is an AP system according to the CAP theorem. For strong consistency, Riak sets bucket type property as strongly consistent. Riak also adopts Amazon Dynamo's tunable availability approach [57]. We have briefly discussed Riak in this section. For further details, a book "A Little Riak Book" by Eric Redmond can be referred [82].

Chapter 6

Other Datastores

NoSQL datastores has variety of classifications. Graph based NoSQL datastores are one among them. Neo4J [25] is a famous Graph datastore. Neo Technology Inc. developed Neo4J and it is available in open source as well as commercial version. Use cases like Tagging, metadata annotation, routing visualization, social networks etc. use Neo4J very well. It is also suitable for distributed storage. Neo4J was developed in Java and supports binding for many languages such as Python, Ruby, Jython, etc.

It is interesting to look at a newly developed datastore known as Cockroach DB. Cockroach DB [15] is a NewSQL datastore. NewSQL datastores combines scalability of NoSQL with ACID guarantees of relational databases together for online transaction processing (OLTP) [77]. It is developed by ex Google employees [58]. Cockroach promises tunable consistency with ACID transaction and indexing support. It has LSM write optimized design. Cockroach DB looks very promising from initial impression but it still needs to make impact in the market.

Chapter 7

Environment and Setup

In this chapter we describe the environment we used and the implementation details regarding benchmarking of different datastores. We describe the cluster, benchmarking tool and technical requirements of the datastores deployment.

7.1 Triton Cluster

Aalto school of science faculties had jointly installed Triton cluster to facilitate the High Performance Computing infrastructure of the schools. Triton cluster is part of the Finnish Grid Infrastructure since 2012. It has several nodes with memory ranging from 16GB to 256GB. A frontend node, HP SL390s G7 with 48GB of memory is used by the user can to access rest of nodes. No external network can access the Triton cluster. Also, Triton has well implemented scheduling concept as nodes are clubbed into partitions.

For our experiment, we have used 5 identical nodes from the cluster with the following hardware specifications shown in the Table 7.1.

Chassis	HP SL390s G7
Networking	Gigabit Ethernet
Processor	2x Intel Xeon X5650 2.67GHz
RAM	48 GB of DDR3-1066 memory
Storage	830 GB of local diskspace

Table 7.1: Hardware of the nodes

7.2 YCSB

Yahoo Cloud Serving Benchmark (YCSB) [54] framework was designed with an aim to smoother the performance comparison of cloud datastores. YCSB is an open source benchmarking framework designed by Yahoo to compare performance of distributed NoSQL datastores like Cassandra, HBase, and PNUTS [53]. The YCSB has six inbuilt core workloads. These workloads have different scenarios by mixing read, write, update and then table scan. A single client is sufficient to run workload on a small cluster (nodes upto ten in number). To run a workload on YCSB, user need to follow these six steps in sequence:

1. Set up the datastore to test:

   If the user is running a cluster, then datastore needs to be installed and set properly prior running the workload. Then key space/ tables/ buckets to store data should be created because client will not automatically do this step for the user.

2. Select the appropriate DB interface layer:

   DB interface layer interacts with the datastore's API and accomplish read, write, update and other operations. Com.yahoo.ycsb.BasicDB is the dummy interface layer of the YCSB.

3. Select the appropriate workload:

   Based on the need and benchmarking requirement, user need to choose appropriate workload from six core workloads of YCSB. Also, YCSB allows the user to create its own workload.

4. Choose the appropriate runtime parameters:

   Parameters like the number of threads, the target number of operations per second and status needs to be set through command line interface while running a workload.

5. Load the data:

   In the loading phase, the client loads the data in datastores via command line commands. For a standard dataset, following command can be used

   ./bin/ycsb load basic –P workloads/workloada

6. Execute the workload:

   After data is loaded, the client can be instructed via command line to run the transaction section. Following command is used to execute the workload:

   ./bin/ycsb run basic –P workloads/workloada –P large.dat -s >transactions.dat

7.3 HBase

For the purpose of this project, we have used Apache HBase open source release 0.96.2. HBase runs on top of Hadoop and use HDFS to store data. We use Apache Hadoop 2.6.0 release. We had installed a five node cluster on Triton identical nodes which we had chosen for our experiment. Within these nodes, we deploy the HBase Master where we deploy the Namenode and a RegionServer along with each DataNode. In the deployment process, we have to fine tune many parameters. Here is the list of few important modifications and the configuration changes we have made:

- *dfs.datanode.handler.count* parameter in Hadoop manages the number of threads that perform data block request from DataNdes. We have set it to six. It will increase memory utilisation of DataNode.
- *dfs.datanode.max.xcievers* parameter is set to 4096. The upper bound of the number of file serve by DataNode at a time is define by this parameter.
- HBASE_ HEAPSIZE defines the amount of allocated heap. To increase performance, we have increased its value from 1000 to 2000.
- In HBase, it is often observed in case of write heavy that write speed is unstable because of block updates by RegionServer. Based on our study from the book "HBase Administration Cookbook" [67] and few other studies [75], we have fine tuned some of the parameters:
  - hbase.regionserver.global.memstore.upperLimit set to 35% .
  - hbase.regionserver.global.memstore.lowerLimit set to 30 % .
  - hbase.hstore.blockingStoreFiles set to 16 instead of 7.
  - hbase.hregion.memstore.block.multiplier set to 8 (default is 2).
  - hbase.hregion.memstore.flush.size is set to default (128MB)

- The HBase has three types of installations: standalone mode, pseudo distributed mode and fully distributed mode. We use fully distributed mode and set the parameter *hbase.cluster.distributed* to true.
- ZooKeeper cluster is a necessary element of distributed HBase cluster. Set HBASE_ MANAGES_ ZK to true.To build a ZooKeeper quorum, *hbase.zookeeper.quorum parameter* needs to be tuned dynamically pointing it to the number of nodes we want in the quorum.
- *zookeeper.session.timeout* is set to 1800000 so that the cluster doesn't go offline quickly.
- *hbase.zookeeper.property.ticktime* is set to 6000.
- ZooKeeper sets the active connection limit. So we have set its value via *hbase.zookeeper.property.maxclientcnxns* to 100.

7.4 MongoDB

For MongoDB deployment, we use the version 2.6.7. MongoDB is available in open source project. MongoDB can be installed in standalone mode or in the form of sharded cluster. For our benchmarking experiment, we have installed five node sharded cluster on Triton. Once we enable sharding for a database, MongoDB allocates primary shard to it and the data is first stored in it before sharding starts. Replica set is the feature of MongoDB to achieve high availability by replicating data across multiple nodes. A replica set contains a primary and few secondary servers. A replica set can have 2–12 mongod instances at a time. In a sharded cluster we have three types of server :

- Config servers that store metadata of the cluster.
- Shard servers that store data subset and replica sets.
- Routing server *mongos*, that routes the operational calls from the client to shard servers

Initially, it is necessary to setup individual instances of replica sets. Then parameters in the configuration file of mongodb (mongodb.config) are modified to include replica set information. When mongos and configuration servers are deployed, then replica sets are added to the cluster. We do need to shard the collections and for this MongoDB has inbuilt function sh.shardCollection

(). We have enabled sharding on datastore Collections which is sharded by _ *key*. We have followed step wise procedure to deploy MongoDB sharded cluster as explained in official MongoDB documentation [17]. For specific technical details, official MongoDB documentation [17] can be referred.

7.5 Cassandra

Apache Cassandra is a project of Apache software foundation and we use its open source version Cassandra 2.0. The Cassandra architecture is a peer–to–peer ring shaped structure. Therefore, we have installed Cassandra on each nodes which we are willing to include in our ring. Snitches are important components of Cassandra deployment process. Each node must have identical snitch configuration.

The main configuration file for Cassandra is the cassandra.yaml file. We have modified the configuration file according to instructions given in Datastax documentation [16]. Here is the list of important consideration that needs to keep in mind while deploying and benchmarking the Cassandra cluster:

- Select few nodes as seed nodes. Do not declare all nodes as seed nodes. Ring nodes take help from seed nodes to find each other and get the information about ring topology. The seed should be configured for each node.
- num_tokens are used for vnodes. We have set its value for each node as 256.
- There is a default seed provider in Cassandra cluster. It maintains a list of IP addresses of the nodes that act as contact points when a new node join the cluster. in the seed_provider value, we need to mention the default seed provider. *org.apache.cassandra.locator.SimpleSeedProvider* is the default seed provider.
- We have created keyspace usertable with replication_factor=1.
- We have created column family data. We have also set its specific parameters as column_type = Standard and comparator = UTF8Type.

7.6 Riak

Basho technologies provides open source version of Riak and we have used Riak version 2.0.2. Basho recommends five node Riak cluster by default.

We have also deployed a five node Riak cluster on Triton. We followed Riak cluster administration describe by Basho technologies documentation [11]. Basho labs has also designed Riak YCSB client to work with YCSB. We have used this for our benchmarking experiments [31].

Chapter 8

Experiment and Evaluation

In this section we benchmark different NoSQL datastores against each other.

Yahoo has developed a test framework known as Yahoo Cloud Servicing Benchmark (YCSB). It is an industry standard benchmark and we used it for our benchmarking experiments. We have discussed different types of NoSQL datastores namely, Key–Value datastores, Document datastores and Extensible Record datastores in earlier sections. These datastores differ in their data models. NoSQL datastores also vary in the way they distribute data among multiple nodes in a cluster. Some distribute data by the range of their keys. Some perform replication of data in the cluster for better availability. Though all datastores provide horizontal scalability but their performance is an important factor.

YCSB evaluates performance of different datastores in multiple scenarios. For this, YCSB creates ***workloads***. The workloads simulate the common operations conducted by the application that use the datastore. Workload is a combination of keys and values. YCSB has six inbuilt workloads [54] [52].

- Workload A: Update heavy workload

  It mixes 50–50 percent of reads and writes.

- Workload B: Read mostly workload

  It has 95% read and 5% write operations.

- Workload C: Read only

  It only has read operations.

- Workload D: Read latest workload

  In this workload, the recently inserted records are more popular.

- Workload E: Short ranges

  This workload query a short range of values instead of particular record.

- Workload E:Read–modify–write

  In this workload, the client read, modify then write back the record to the datastore.

YCSB uses its own data for the benchmarking experiments. This data is illustrated as table of records. YCSB is so flexible that the user can choose number of records, number of fields and size of records based on its requirement.

In our experiment, we use 100 million records each of size 1KB. Every record has 10 fields and each field is of 100 Bytes. Total Size of the RAM on Triton cluster nodes is sufficient enough to accumulate all the data (48GB RAM on each node). YCSB already has support for HBase, Cassandra and MongoDB [52]. For Riak, Basho Technologies has developed Riak Client for YCSB [31]. We analyzed the performance of these five datastores using YCSB.

Below we present some of the benchmark results in different scenarios.

8.1 Load Phase

In the load phase, we have uploaded 100 million records in every datastore. MongoDB has high latency compared to all with its throughput ranging around 9000 operations per second. The Figure 8.1 depicts the latency versus throughput comparison of different datastores. Riak demonstrates the lowest throughput and Cassandra is the clear winner compared to others in terms of throughput. This is because Cassandra updates the data in memory and writes it to the disk at the same time. HBase has throughput of 14000 operations/second and lowest average latency. HBase has a write optimized design which helps it to provide good performance in case of heavy load.

8.2 Mixed Reads and Updates

In this operation, we mainly focus on update. While loading 100 million records, 50% is update and 50% is read operations. The Figure 8.2 describes the latency/throughput curve for different datastores. Cassandra and HBase demonstrate excellent results with average latency below 5ms. HBase has the lowest latency because of its write optimized design. HBase does not

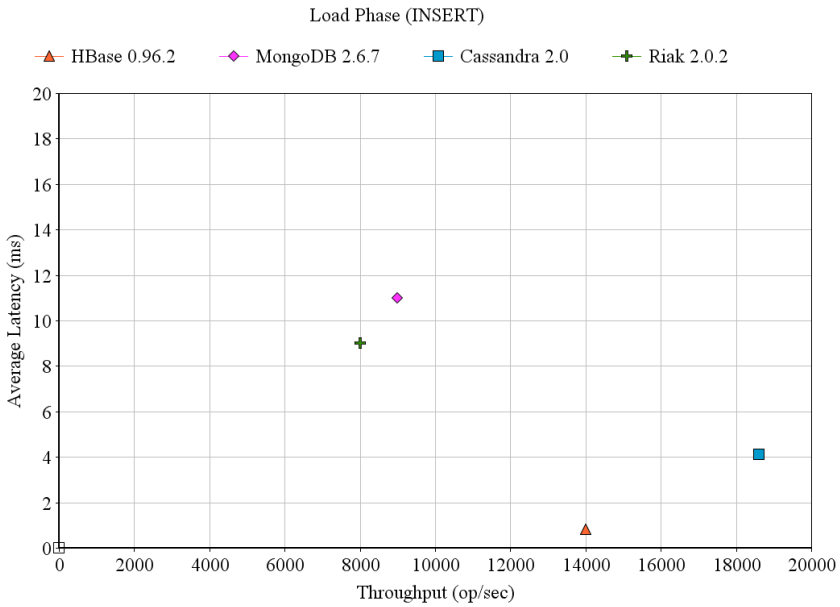

Figure 8.1: Throughput v/s Latency

directly sync updates to the disk. Google's BigTable is the mother of HBase and Cassandra data model. Throughput of Cassandra was constantly increasing and when we kept running the workload, Cassandra finally achieved highest throughput. The graph only contain initial deviation of performance. MongoDB has poor performance for this workload but Riak show considerably stable performance than MongoDB.

8.3 Read Heavy Workload

Read heavy workload has 95% reads and just 5% updates. In YCSB this is known as *Workload B*. Upon analyzing the results we conclude that MongoDB is the clear winner in read operations. MongoDB supports memory mapped files for caching which give it an clear edge over others in read performance. HBase has bad read performance with highest average latency. However, its throughput remains high. Key–row caching of Cassandra helps it to give a much better read performance. Riak shows better read performance than HBase. The Figure 8.3 shows the complete scenario.

On the other hand, in case of update operation of this workload, HBase clearly wiped out its competitors. The Figure 8.4 illustrates the update operation results for read heavy workload. Cassandra closely follows HBase

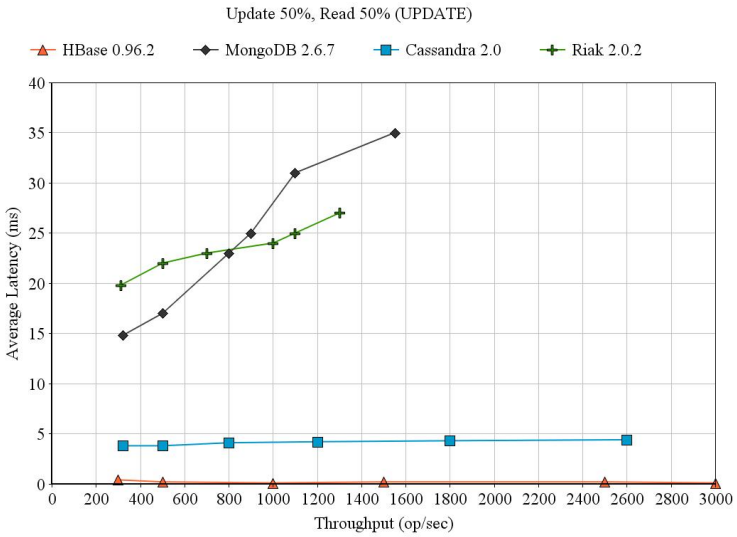

Figure 8.2: Workload 50% read 50% update –Update

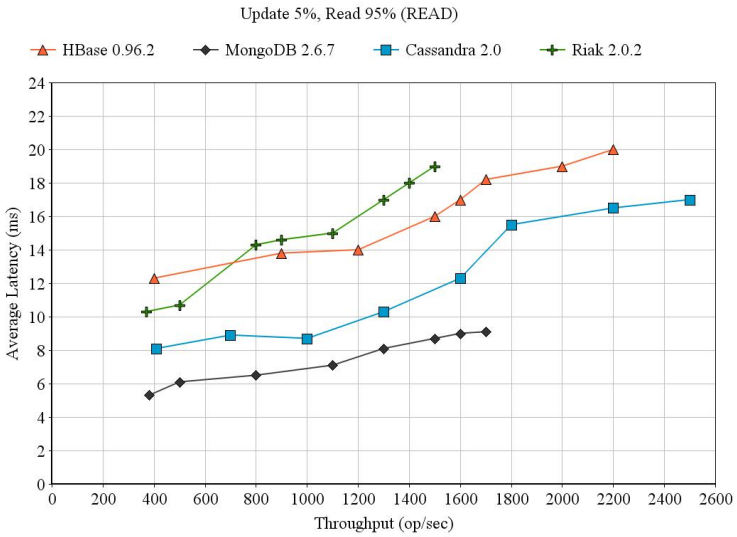

Figure 8.3: Workload 95% read 5% update –Read

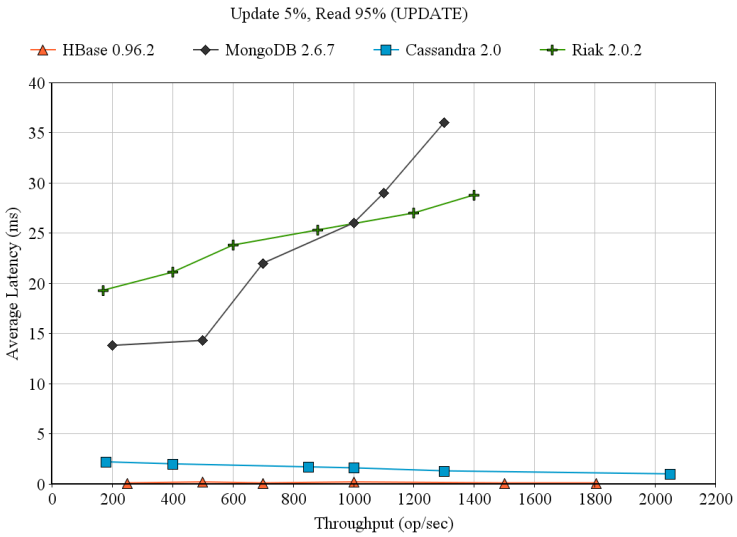

Figure 8.4: Workload 95% read 5% update –Update

in updates with very less latency. HBase first commits the edits to memstores and then it flush the aggregated edits to the HLog in asynchronous manner. Cassandra also has similar approach where writes are first written to commit log. These features make HBase and Cassandra a clear winner for updates. Riak was better than MongoDB in this case.

8.4 Read Only

The read only workload contains 100% read operations. The Figure 8.5 shows the latency/throughput curve of different datastores for read only workload. MongoDB performance is still much better than the rest. Cassandra also shows good performance whereas HBase shows poor read performance. Riak is bit better than HBase in this category.

8.5 Insert Mostly

This workload mostly consists of insert operations. As we have seen in earlier graphs, HBase and Cassandra are the winners. They show very low latency and high throughput. MongoDB performs worst and has very high latency. Riak performs much better than MongoDB with average latency ranging around 6ms. The Figure 8.6 shows the insert mostly workload results.

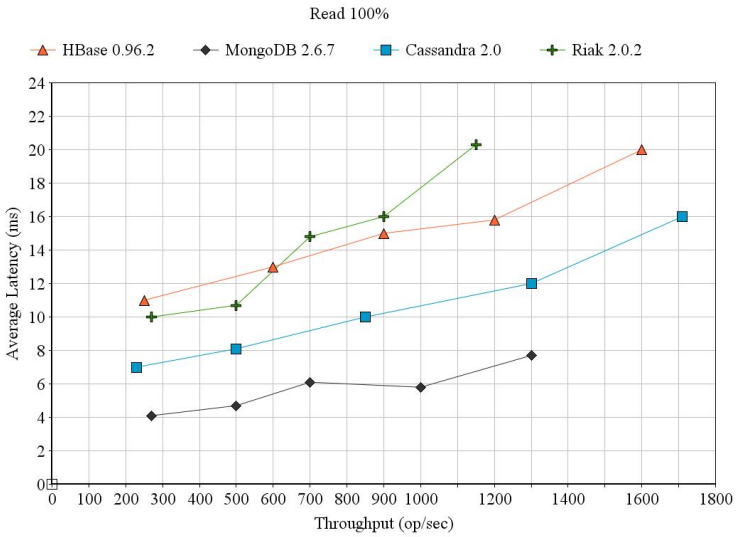

Figure 8.5: Read 100%

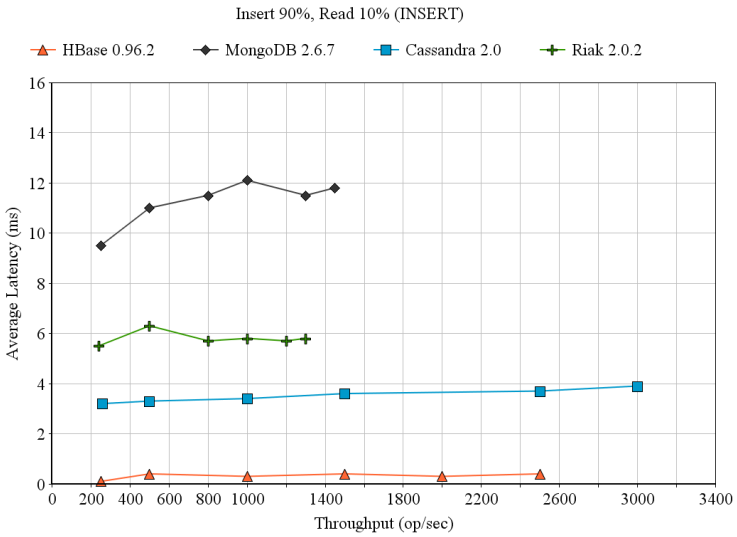

Figure 8.6: Insert 90% read 10% –Insert

Chapter 9

Conclusions

In this chapter we review work done in the thesis and present our viewpoints on further improvements in the section 8.

9.1 Discussion

This thesis presents a holistic overview of different NoSQL datastores available to store Big Data. Database technologies have evolved according to the business needs. For many decades, traditional relational database management systems (RDBMS) were primary choice to store data. They use relational data model. The relational database is divided into tables and they are connected through "key field". RDBMS maintain data in tables and relationships which are created among data and tables. RDBMS are still popular choice to store structured data.

Digital world has grown significantly in 21st century. Data is generated from different sources and it is in both structured and unstructured form. To process and store huge amount of data, companies do need modern technologies. RDBMS face challenges in scaling while dealing with large volume of data. NoSQL datastores have evolved to address the limitations of RDBMS. The big companies like Amazon, Google, Facebook and Yahoo decided to develop new technologies to address their business needs related to process and store immense amount of online data. The development of MapReduce paradigm, BigTable, Google File System, Amazon Dynamo laid initial foundation for the NoSQL datastores. Variety of NoSQL datastores have been introduced in the market in last few years. We have surveyed few of them in this thesis.

NoSQL datastores are broadly classified into three categories. These are: Key–Value datastore, Extensible Record datastore and Document datastores.

We have looked deeply into few examples of all three classifications of NoSQL datastores. From Document datastores, we have covered MongoDB. MongoDB is a scalable datastore and it is easily scalable within and across different data centers. MongoDB has a pluggable storage architecture. We have discussed scalable environment provided by MongoDB architecture. MongoDB has inbuilt capability to horizontally scale out its architecture for low cost community hardware using sharding. MongoDB also provides native replication and automated failover which gives high reliability and operational exibility. MongoDB is a CP datastore but with few bugs [1] [2]. A blog post [13] proves that MongoDB's consistency model is broken by design. It is also observed by them that the MongoDB sometimes read stale version of documents and can also show garbage data from non occurring writes. These days MongoDB finds its acceptability in variety of industries like banking, e–commerce, insurance and telecommunication.

We have further discussed HBase as an example of Column Oriented datastores. Google's Big Table is the motivation for HBase design. HBase runs on the top of HDFS. We have also looked into its data model in detail. HBase does not support multi–row ACID transactions like RDBMS but individual row operations are atomic. These operations are supported by row level locking and transactions. Fault Tolerance is a key aspect of HBase design. It uses ZooKeeper as coordination service and failure detection component. HBase is a CP datastore according to the CAP theorem.

Cassandra and Riak were two examples we took from Key–Value datastores. Cassandra has ring shape peer–to–peer architecture. Unlike relational management databases, Cassandra does not require matching columns within a row. Its decentralised architecture avoids single point of failure and any node can serve the request. The Riak cluster also has ring shaped architecture. Riak can be assumed as a very large hash space. Here, the value is hashed into specific location using a unique hashing key and it can be retrieve anytime. Cassandra and Riak both are AP datastores as per the CAP theorem.

Due to different data models and architecture design considerations we have seen, these datastores differ in their performance. We have benchmarked these four datastores against each other using YCSB framework to analyze their performance on a distributed cluster. From our experiments we conclude that due to write optimized design of HBase and Cassandra, they showed very good performance in heavy load of data and update operations.

[1] MongoDB bug: https://jira.mongodb.org/browse/SERVER-17975 (accessed: 11/05/2015)

[2] MongoDB bug: https://jira.mongodb.org/browse/SERVER-18022 (accessed: 11/05/2015)

But HBase has poor read performance. Even Riak was more stable than HBase in read operations. MongoDB shown edge over others in read performance. We can further conclude that companies should not blindly opt for any NoSQL datastore for scalability of their business operations. Based on the specific needs and business requirements, companies have to make trade–off between the specific properties they need for their database.

9.2 Future Work

The thesis has covered few NoSQL datastores and their performance aspects. There is still scope for further considerations. We have used default behavior of HBase for our experiments. HBase version with pre–split regions can be used for future benchmarking experiments. We hope it will further give better throughput during load operations. MongoDB Inc. has released MongoDB version 3.0 recently [4]. The company is promising huge boost in performance compared to its version 2.6.X, therefore it would be a good choice to use MongoDB version 3.0 for future benchmarking experiments. However, the bugs found by Aphyr in MongoDB's consistency model are still open [33]. We have not covered other NoSQL datastores like Redis, Voldemort, and DynamoDB. Future works may take these datastores also into consideration.

It is a fact that NoSQL datastores provide high scalability and can manage huge amount of data. But not every company is Google, Amazon, or Facebook. Many companies do not need to manage that huge amount of data but do need very good read and write performances. MySQL cluster provide limited scalability. Thesis leaves scope to benchmark MySQL cluster against NoSQL datastores. Cockroach DB is another datastore that we have briefly touched in this thesis. It is a NewSQL datastore and it would be interesting to see its performance against NoSQL datastores.

Bibliography

[1] Amazon EC2. http://aws.amazon.com/ec2/ (Accessed: 02/01/2015).

[2] Amazon Inc. Amazon Web Service web page. https://www.aws.amazon.com (Accessed: 05/03/2015).

[3] Amazon S3. http://aws.amazon.com/s3/ (Accessed: 02/01/2015).

[4] Announcing MongoDB 3.0. http://www.mongodb.com/blog/post/announcing-mongodb-30 (Accessed: 02/04/2015).

[5] Apache Cassandra. http://cassandra.apache.org/ (Accessed: 02/03/2015).

[6] Apache Hadoop. https://hadoop.apache.org/ (Accessed: 02/01/2015).

[7] Apache HBase. http://hbase.apache.org/ (Accessed: 02/01/2015).

[8] Apache HBase I/O HFile. http://blog.cloudera.com/blog/2012/06/hbase-io-hfile-input-output/ (Accessed: 02/05/2015).

[9] Apache Hive TM. https://hive.apache.org/ (Accessed: 02/03/2015).

[10] Apache Zookeeper. https://zookeeper.apache.org/ (Accessed: 02/03/2015).

[11] Basho Technologies. http://basho.com/ (Accessed: 02/04/2015).

[12] Call me maybe: MongoDB. https://aphyr.com/posts/284-call-me-maybe-mongodb (Accessed: 05/05/2015).

[13] Call me maybe: MongoDB stale reads. https://aphyr.com/posts/322-call-me-maybe-mongodb-stale-reads (Accessed: 02/05/2015).

[14] CAP Theorem. http://www.abramsimon.com/cap-theorem/ (Accessed: 05/05/2015).

[15] Cockroach DB. http://cockroachdb.org/ (Accessed: 05/03/2015).

[16] Datastax Documentation. http://docs.datastax.com/en/cassandra/1.2/cassandra/configuration/configCassandra_yaml_r.html#reference_ds_qfg_n1r_1k__endpoint_snitch (Accessed: 02/05/2015).

[17] Deploy a Sharded Cluster. http://docs.mongodb.org/manual/tutorial/deploy-shard-cluster/ (Accessed: 12/04/2015).

[18] Dropbox Fact Sheet. https://www.dropbox.com/static/docs/DropboxFactSheet.pdf (Accessed: 02/03/2015).

[19] Dropbox Inc. https://www.dropbox.com/ (Accessed: 02/03/2015).

[20] Heroku Inc. https://www.heroku.com (Accessed: 05/03/2015).

[21] Microsoft Inc. Microsoft Azure web page. https://www.azure.microsoft.com (Accessed: 05/03/2015).

[22] Mongodb Documentation Project. http://docs.mongodb.org/v2.6/MongoDB-manual.pdf (Accessed: 12/05/2015).

[23] Mongodb Inc. http://www.mongodb.com/ (Accessed: 02/03/2015).

[24] MySQL Cluster CGE. https://www.mysql.com/products/cluster/ (Accessed: 02/01/2015).

[25] Neo4J Database. http://neo4j.com/ (Accessed: 12/05/2015).

[26] NoSQL–A Relational Database Management System. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page/ (Accessed: 02/05/2015).

[27] OpenNebula. Flexible Enterprise Cloud Made Simple. http://opennebula.org/ (Accessed: 02/01/2015).

[28] OpenShift. https://www.openshift.com (Accessed: 05/03/2015).

[29] Openstack Compute. http://www.openstack.org/software/openstack-compute/ (Accessed: 02/01/2015).

[30] Oracle Real Application Clusters. http://www.oracle.com/us/products/database/options/real-application-clusters/overview/index.html (Accessed: 02/01/2015).

[31] Riak Client for YCSB. `https://github.com/basho-labs/YCSB/tree/master/riak` (Accessed: 01/04/2015).

[32] Salesforce Inc. `https://www.saleforce.com` (Accessed: 05/03/2015).

[33] Stale Reads with WriteConcern Majority and ReadPreference Primary. `https://jira.mongodb.org/browse/SERVER-17975` (Accessed: 02/03/2015).

[34] Welcome to Apache Pig! `https://pig.apache.org/` (Accessed: 02/03/2015).

[35] Welcome to Flickr- Photo Sharing. `https://www.flickr.com/` (Accessed: 02/04/2015).

[36] What is the Hadoop Distributed File System (HDFS)? `http://www-01.ibm.com/software/data/infosphere/hadoop/hdfs/`(Accessed: 05/03/2015).

[37] MD5 Hash Function. In *Encyclopedia of Cryptography and Security, 2nd Ed.* 2011, p. 771.

[38] Aiyer, A. S., Bautin, M., Chen, G. J., Damania, P., Khemani, P., Muthukkaruppan, K., Ranganathan, K., Spiegelberg, N., Tang, L., and Vaidya, M. Storage Infrastructure Behind Facebook Messages: Using HBase at Scale. *IEEE Data Eng. Bull. 35*, 2 (2012), 4–13.

[39] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. A View of Cloud Computing. *Commun. ACM 53*, 4 (2010), 50–58.

[40] Barrett, L. SaaS Market Growing by Leaps and Bounds, July 2010. `http://www.datamation.com/entdev/article.php/3895101/SaaS-Market-Growing-by-Leaps-and-Bounds-Gartner.htm` (Accessed: 02/03/2015).

[41] Bershad, B. N., and Mogul, J. C., Eds. *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA* (2006), USENIX Association.

[42] Bloom, B. H. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM 13*, 7 (1970), 422–426.

[43] Brewer, E. A. Towards Robust Distributed Systems (abstract). In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, July 16-19, 2000, Portland, Oregon, USA.* (2000), p. 7.

[44] Brewer, E. A. Pushing the CAP: Strategies for Consistency and Availability. *IEEE Computer 45*, 2 (2012), 23–29.

[45] Burrows, M. The Chubby Lock Service for Loosely-Coupled Distributed Systems. In Bershad and Mogul [41], pp. 335–350.

[46] Cattell, R. Scalable SQL and NoSQL data stores. *SIGMOD Record 39*, 4 (2010), 12–27.

[47] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst. 26*, 2 (2008).

[48] Chodorow, K., and Dirolf, M. *MongoDB - The Definitive Guide: Powerful and Scalable Data Storage.* O'Reilly, 2010.

[49] Codd, E. F. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM 13*, 6 (1970), 377–387.

[50] Codd, E. F. Relational Database: A Practical Foundation for Productivity. *Commun. ACM 25*, 2 (1982), 109–117.

[51] Constine, J. Congratulations Crunchies Winners! Dropbox Is The Best Overall Startup, January 2012. `http://techcrunch.com/2012/01/31/crunchies-dropbox/` (Accessed: 12/05/2015).

[52] Cooper, B. F. Running a Workload. `https://github.com/brianfrankcooper/YCSB/wiki/Running-a-Workload` (Accessed: 05/05/2015).

[53] Cooper, B. F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H., Puz, N., Weaver, D., and Yerneni, R. PNUTS: py.

[54] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, Indianapolis, Indiana, USA, June 10-11, 2010* (2010), pp. 143–154.

[55] de la Iglesia, J. L. M., and Gayo, J. E. L. Doing Business by Selling Free Services. In *Web 2.0*. Springer, 2009, pp. 1–14.

[56] Dean, J., and Ghemawat, S. Mapreduce: Simplified Data Processing on Large Clusters. *Commun. ACM 51*, 1 (2008), 107–113.

[57] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. Dynamo: Amazon's Highly Available Key-Value Store. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007* (2007), pp. 205–220.

[58] Finley, K. Out in the Open: Ex Googlers Building Cloud Software That's Almost Impossible To Take Down, July 2014. `http://www.wired.com/2014/07/cockroachdb/` (Accessed: 02/03/2015).

[59] Ganesh, A. J., Kermarrec, A., and Massoulié, L. Peer-to-Peer Membership Management for Gossip-Based Protocols. *IEEE Trans. Computers 52*, 2 (2003), 139–149.

[60] George, L. *HBase - The Definitive Guide: Random Access to Your Planet-Size Data*. O'Reilly, 2011.

[61] Gilbert, S., and Lynch, N. A. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. *SIGACT News 33*, 2 (2002), 51–59.

[62] Gray, J., and Reuter, A. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[63] Hewitt, E. *Cassandra - The Definitive Guide: Distributed Data at Web Scale*. Springer, 2011.

[64] Huang, C., Hu, W., Shih, C. C., Lin, B., and Cheng, C. The Improvement of Auto-Scaling Mechanism for Distributed Database - A Case Study for MongoDB. In *15th Asia-Pacific Network Operations and Management Symposium, APNOMS 2013, Hiroshima, Japan, September 25-27, 2013* (2013), pp. 1–3.

[65] Hudak, P. Conception, Evolution, and Application of Functional Programming Languages. *ACM Comput. Surv. 21*, 3 (1989), 359–411.

[66] Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. Zookeeper: Wait-free Coordination for Internet-scale Systems. In *2010 USENIX Annual Technical Conference, Boston, MA, USA, June 23-25, 2010* (2010).

[67] Jiang, Y. *HBase Administration Cookbook*. Packt Publishing Ltd, 2012.

[68] Karger, D. R., Lehman, E., Leighton, F. T., Panigrahy, R., Levine, M. S., and Lewin, D. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997* (1997), pp. 654–663.

[69] Lakshman, A., and Malik, P. Cassandra: A Structured Storage System on a P2P Network. In *SPAA 2009: Proceedings of the 21st Annual ACM Symposium on Parallelism in Algorithms and Architectures, Calgary, Alberta, Canada, August 11-13, 2009* (2009), p. 47.

[70] Lakshman, A., and Malik, P. Cassandra: A Decentralized Structured Storage System. *Operating Systems Review 44*, 2 (2010), 35–40.

[71] Lämmel, R. Google's Mapreduce Programming Model –Revisited. *Sci. Comput. Program. 70*, 1 (2008), 1–30.

[72] Lamport, L. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM 21*, 7 (1978), 558–565.

[73] Mell, P., and Grance, T. The NIST Definition of Cloud Computing.

[74] Merkle, R. C. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings* (1987), pp. 369–378.

[75] MEYN, O. Optimizing Writes in Hbase, July 2012. `http://gbif.blogspot.fi/2012/07/optimizing-writes-in-hbase.html` (Accessed: 02/05/2015).

[76] Michael, M. M., Moreira, J. E., Shiloach, D., and Wisniewski, R. W. Scale-up x Scale-out: A Case Study using

Nutch/Lucene. In *21th International Parallel and Distributed Processing Symposium (IPDPS 2007), Proceedings, 26-30 March 2007, Long Beach, California, USA* (2007), pp. 1–8.

[77] Moniruzzaman, A. B. M. NewSQL: Towards Next-Generation Scalable RDBMS for Online Transaction Processing (OLTP) for Big Data Management. *CoRR abs/1411.7343* (2014).

[78] Moore, G. E., et al. Cramming More Components onto Integrated Circuits. *Proceedings of the IEEE 86*, 1 (1998), 82–85.

[79] O'Neil, P. E., Cheng, E., Gawlick, D., and O'Neil, E. J. The Log–Structured Merge–Tree (LSM–Tree). *Acta Inf. 33*, 4 (1996), 351–385.

[80] Pritchett, D. BASE: An ACID Alternative. *ACM Queue 6*, 3 (2008), 48–55.

[81] Ramakrishnan, R., and Gehrke, J. *Database Management Systems (3. ed.)*. McGraw-Hill, 2003.

[82] Redmond, E., and Daily, J. *A Little Riak Book 2.0*. Basho, 2014.

[83] Schollmeier, R. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In *1st International Conference on Peer-to-Peer Computing (P2P 2001), 27-29 August 2001, Linköping, Sweden* (2001), pp. 101–102.

[84] Scott, M. L., and Peterson, L. L., Eds. *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003* (2003), ACM.

[85] Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M., et al. A Comprehensive Study of Convergent and Commutative Replicated Data Types.

[86] Shapiro, M., Preguiça, N. M., Baquero, C., and Zawirski, M. Conflict-Free Replicated Data Types. In *Stabilization, Safety, and Security of Distributed Systems - 13th International Symposium, SSS 2011, Grenoble, France, October 10-12, 2011. Proceedings* (2011), pp. 386–400.

[87] Shastry, K., Madhyastha, S., Kumar, S., Bresniker, K., and Battas, G. Transaction Support for HBase. In *20th International Conference on Management of Data, COMAD 2014, Hyderabad, India, December 17-19, 2014.* (2014), pp. 117–120.

[88] Silberschatz, A., Korth, H. F., and Sudarshan, S. *Database System Concepts, 5th Edition*. McGraw-Hill Book Company, 2005.

[89] Sutter, H. The Free Lunch is Over: A Fundamental Turn Toward Concurrency in Software.

[90] Velte, T., Velte, A., and Elsenpeter, R. *Cloud Computing, A Practical Approach*, 1 ed. McGraw-Hill, Inc., New York, NY, USA, 2010.

[91] Vogels, W. Eventually Consistent. *Commun. ACM 52*, 1 (2009), 40–44.

[92] Wang, G., and Tang, J. The NoSQL Principles and Basic Application of Cassandra Model. In *Proceedings of the 2012 International Conference on Computer Science and Service System* (Washington, DC, USA, 2012), CSSS '12, IEEE Computer Society, pp. 1332–1335.

[93] Wang, X., and Loguinov, D. Load-Balancing Performance of Consistent Hashing: Asymptotic Analysis of Random Node Join. *IEEE/ACM Trans. Netw. 15*, 4 (2007), 892–905.

[94] Welsh, M., Culler, D. E., and Brewer, E. A. SEDA: An Architecture for Well-Conditioned, Scalable Internet Services. In *SOSP* (2001), pp. 230–243.